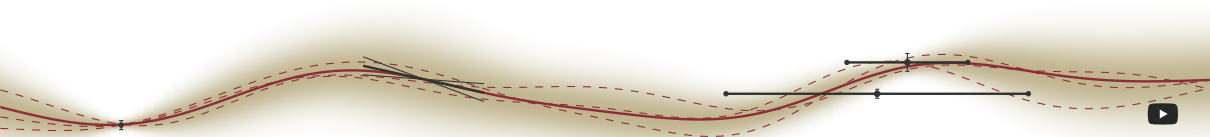
Numerics of Machine Learning
Lecture 01
Introduction

Philipp Hennig

20 October 2022

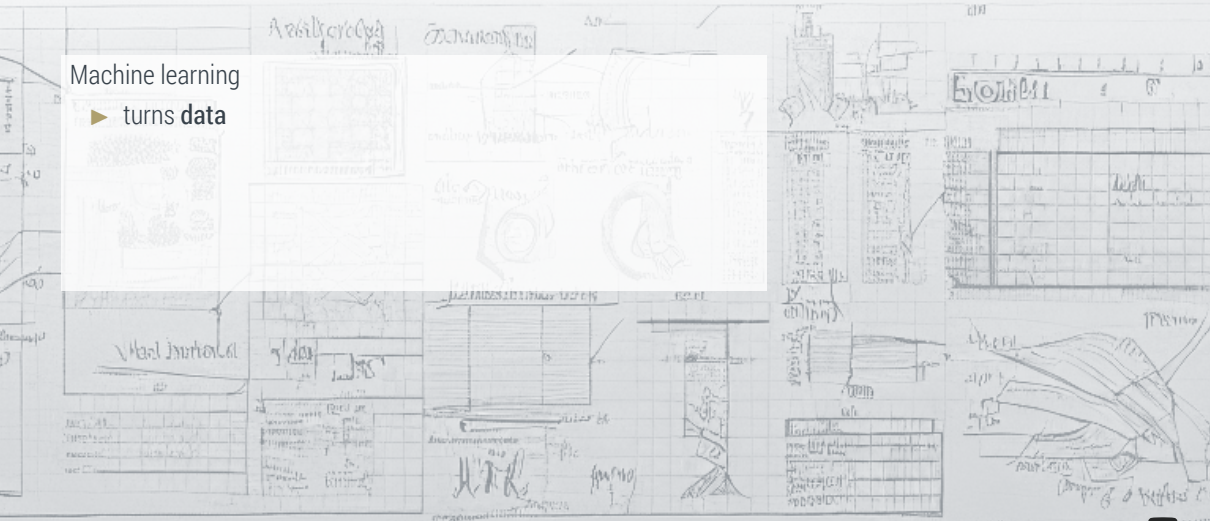
Eberhard Karls
Universität
Tübingen

Faculty of Science
Department of Computer Science
Chair for the Methods of Machine Learning



Why this Class?

Numerical computation is what actually happens inside of a computer

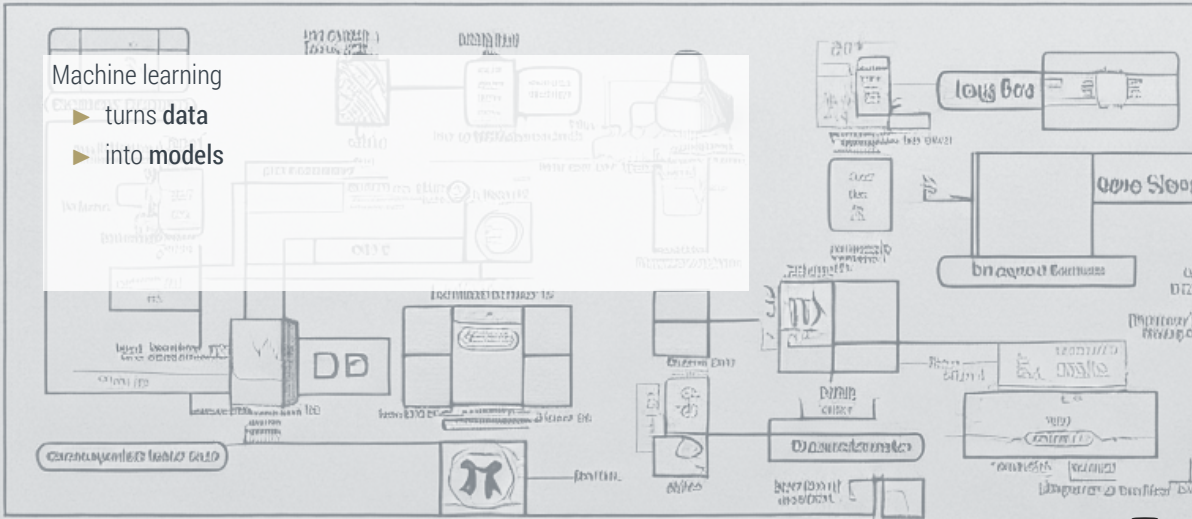


Machine learning

- turns **data**

Why this Class?

Numerical computation is what actually happens inside of a computer



Machine learning

- turns **data**
- into **models**

Why this Class?

Numerical computation is what actually happens inside of a computer

Machine learning

- turns **data**
- into **models**
- using **computation**

Why this Class?

Numerical computation is what actually happens inside of a computer

Machine learning

- turns **data**
- into **models**
- using **computation**

In contrast to **classic AI** (rule-based), in **contemporary ML**, the computation is always a **numerical** computation

Numerical Computations

approximating numbers that can't be exactly computed

(vague) Definition: A **numerical computation** aims to find the answer to a mathematical question that does not have a *tractable, analytic* answer.

attempt at a technical definition: Consider a function $f : \mathcal{X} \to \mathbb{R}$ from inputs $x \in \mathcal{X}$ to the Reals. An algorithm a is a map $a(x) = \hat{y}$ that produces an approximation $\hat{y} \approx f(x)$. If there are values $x_c \in \mathcal{X}$ accepted by a without throwing an exception such that $|a(x_c) - f(x_c)| \gg \varepsilon$, where ε is machine precision, we might call a a *numerical method*. In contrast, algorithms which always produce exact answers could be called *atomic*. In other words, a numerical method is an algorithm that can go wrong, because its task is hard.

Intuition: Atomic operations can be found in `libc`, in base `numpy`, or in `scipy.special`:

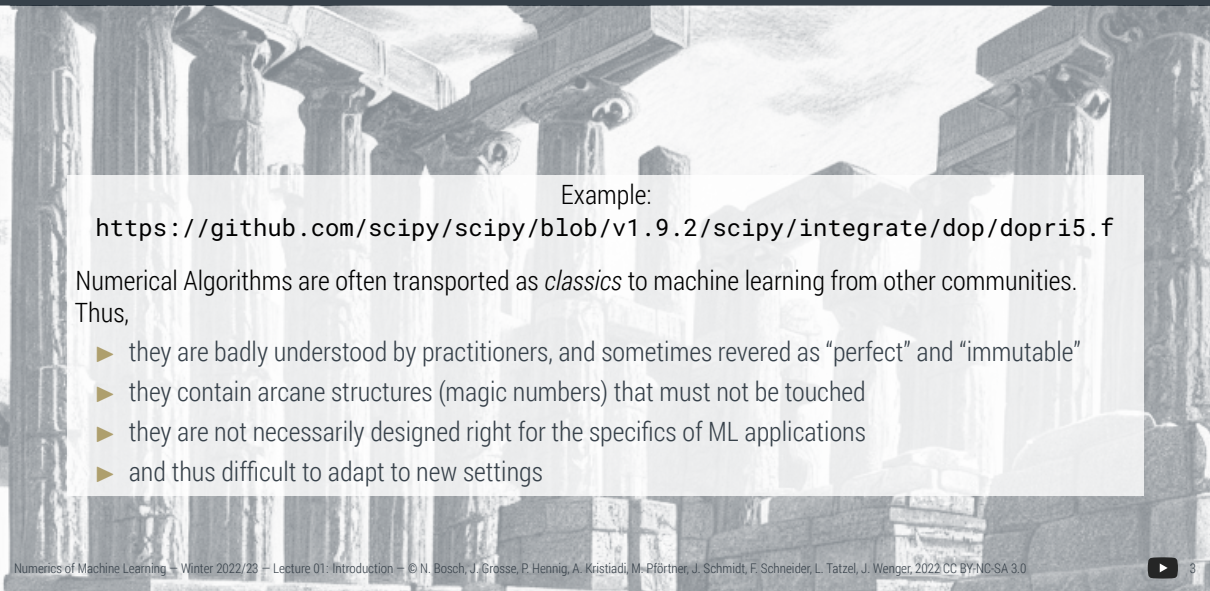
`np.exp`, `np.sin` `sp.special.erfc` `sp.special.gamma`

While *numerical* methods tend to have their own library:

`sp.sparse.linalg.cg` `sp.optimize.minimize` `jax.experimental.ode`

Gifts from the Elders

Classic knowledge should be honoured, but critically reflected



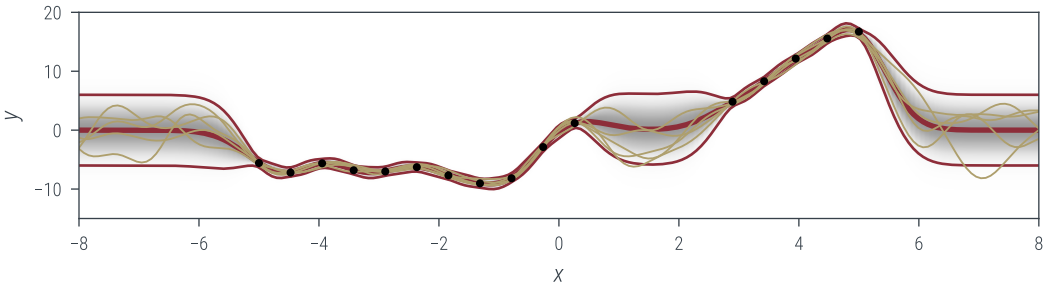
Example:

`https://github.com/scipy/scipy/blob/v1.9.2/scipy/integrate/dop/dopri5.f`

Numerical Algorithms are often transported as *classics* to machine learning from other communities. Thus,

- they are badly understood by practitioners, and sometimes revered as “perfect” and “immutable”
- they contain arcane structures (magic numbers) that must not be touched
- they are not necessarily designed right for the specifics of ML applications
- and thus difficult to adapt to new settings

So what *are* the numerical tasks that drive machine learning?
And why do we perhaps need a new look at them?



From your Probabilistic ML lecture: Regression on a function $f : \mathbb{X} \to \mathbb{R}$, from $p(\boldsymbol{y} \mid f(\boldsymbol{x})) = \mathcal{N}(\boldsymbol{y}; f(\boldsymbol{x}), \sigma^2 I)$ with $\boldsymbol{y} \in \mathbb{R}^n$ and $p(f) = \mathcal{GP}(f; 0, k)$ yields

$$p(f \mid \boldsymbol{y}, \boldsymbol{x}) = \mathcal{GP}(f; m(x_*), v(x_*, x^*)) \quad \text{where}$$

$$m(x_*) = \underbrace{k_{x_*,\boldsymbol{x}}}_{\in \mathbb{R}^{|x_*| \times n}} \underbrace{(k_{\boldsymbol{x},\boldsymbol{x}} + \sigma^2 I)^{-1}}_{\in \mathbb{R}^{n \times n}} \boldsymbol{y} \quad \text{and} \quad v(x_*, x^*) = k_{x_*,x^*} - k_{x_*,\boldsymbol{x}}(k_{\boldsymbol{x},\boldsymbol{x}} + \sigma^2 I)^{-1} k_{\boldsymbol{x},x^*}$$

$$m(x_*) = \underbrace{k_{X_*,\boldsymbol{x}}}_{\in\mathbb{R}^{|x_*|\times n}} \underbrace{(k_{\boldsymbol{x},\boldsymbol{x}} + \sigma^2 I)^{-1}}_{\in\mathbb{R}^{n\times n}} \boldsymbol{y}$$

This is an instance of **least-squares** fitting. You have probably learned it can be solved as

```python
import numpy as np
from scipy.linalg import cho_factor, cho_solve

kXX = k(X, X)
G = kXX + sigma**2 * np.eye(n)
G = cho_factor(G)

kxX = k(x, X)
A = cho_solve(G, Y)
m = kxX @ A
```

from `https://github.com/scipy/scipy/blob/v1.9.2/scipy/linalg/_decomp_cholesky.py`

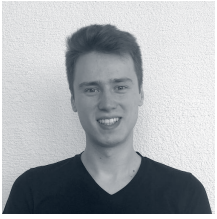
Linear Algebra

The very foundations – least-squares fitting

- you may have been told that "inverting a matrix costs $\mathcal{O}(n^3)$"
- and thus "GP inference & kernel methods do not scale, use deep learning instead!"
- but the truth is *way more complicated!*

In Lectures 1,2,3, we will see

- structure in $\boldsymbol{x}$ can be used to *drastically* (up to exponentially) speed up linear algebra
- if you accept imprecise computations, cost is never cubic to begin with
- Of course it isn't, because $m(x_*)$ is *the minimum of a quadratic function*. It's *easier* than deep learning!

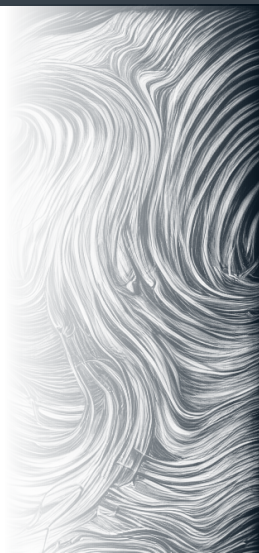


Marvin Pförtner

Jonathan Wenger

Simulation

Predict what will happen next, model dynamical systems



Simulation methods estimate the trajectory of a dynamical system. Roughly, the solution $x(t) : \mathbb{R} \to \mathbb{R}^d$ to an *implicit* equation of the form

$$D(x(t)) = f(x(t))$$

In particular

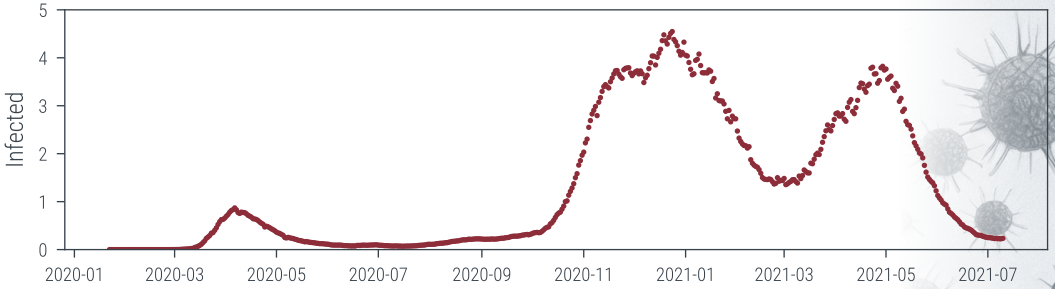
- *ordinary differential equations* ($x'(t) = f(x(t))$)
- *partial differential equations* (D is a differential operator)
- *differential algebraic equations* ($F(x'(t), x(t)) = 0$)

In machine learning, these equations show up

- in **reinforcement learning, control, robotics** to *predict* future dynamics (so the agent can adapt its behaviour)
- in **scientific machine learning** in form of the *laws of nature* providing *mechanistic information*

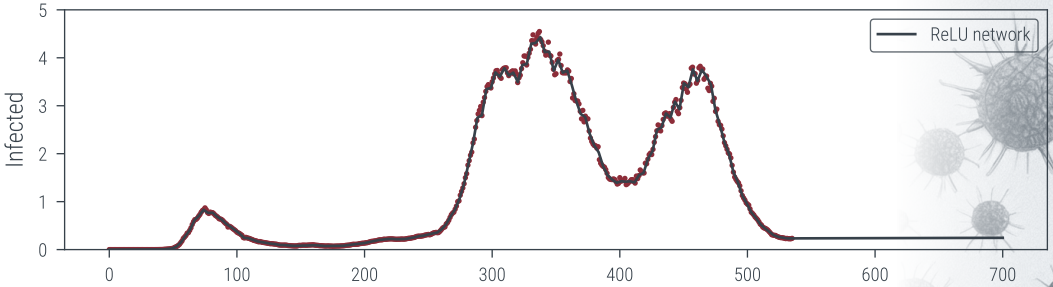
Simulation

Predict what will happen next, model dynamical systems



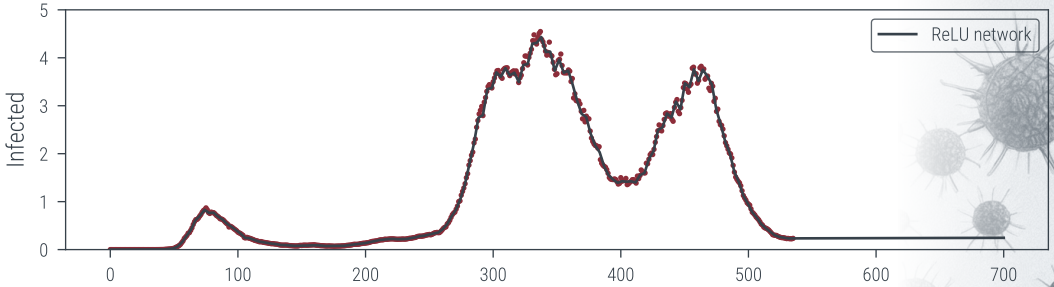
Simulation

Predict what will happen next, model dynamical systems



Simulation

Predict what will happen next, model dynamical systems

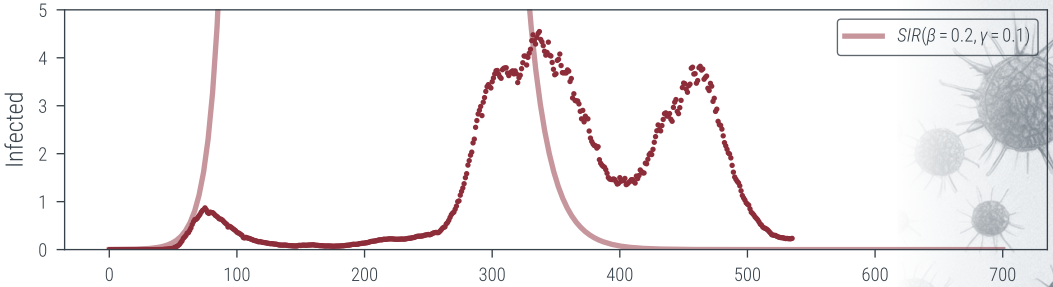


Mechanistic knowledge encodes crucial scientific insight

$$\frac{d}{dt}\begin{bmatrix} S(t) \\ I(t) \\ R(t) \end{bmatrix} = \begin{bmatrix} -\beta \cdot S(t)I(t)/P \\ \beta \cdot S(t)I(t)/P - \gamma I(t) \\ \gamma I(t) \end{bmatrix}$$

Simulation

Predict what will happen next, model dynamical systems



Mechanistic knowledge encodes crucial scientific insight

$$\frac{d}{dt}\begin{bmatrix} S(t) \\ I(t) \\ R(t) \end{bmatrix} = \begin{bmatrix} -\beta \cdot S(t)I(t)/P \\ \beta \cdot S(t)I(t)/P - \gamma I(t) \\ \gamma I(t) \end{bmatrix}$$

But it requires *latent forces* that may be unknown

Simulation

Predict what will happen next, model dynamical systems

How do we infer $\beta(t)$ from the observations?

$$\frac{d}{dt}\begin{bmatrix} S(t) \\ I(t) \\ R(t) \end{bmatrix} = \begin{bmatrix} -\beta(t) \cdot S(t)I(t)/P \\ \beta(t) \cdot S(t)I(t)/P - \gamma I(t) \\ \gamma I(t) \end{bmatrix}$$

- Let's *parametrize* $\beta(t) = \beta(t, \theta)$, e.g. with a deep net
- We could use a classic method, but those aren't differentiable
- luckily, someone implemented them in `jax`. But hang on…

Jonathan Schmidt

Nathanael Bosch

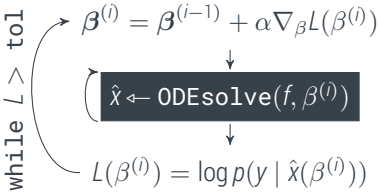
Simulation

Predict what will happen next, model dynamical systems

How do we infer $\beta(t)$ from the observations?

$$\frac{d}{dt}\begin{bmatrix} S(t) \\ I(t) \\ R(t) \end{bmatrix} = \begin{bmatrix} -\beta(t) \cdot S(t)I(t)/P \\ \beta(t) \cdot S(t)I(t)/P - \gamma I(t) \\ \gamma I(t) \end{bmatrix}$$

- Let's *parametrize* $\beta(t) = \beta(t, \theta)$, e.g. with a deep net
- We could use a classic method, but those aren't differentiable
- luckily, someone implemented them in `jax`. But hang on…



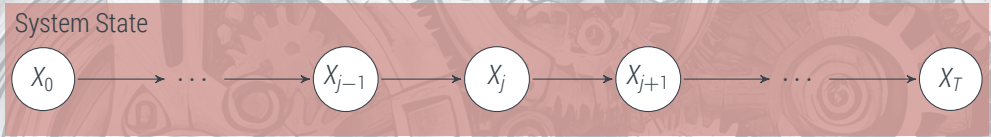
There is a **hidden loop nested** inside our training loop!

Jonathan Schmidt

Nathanael Bosch

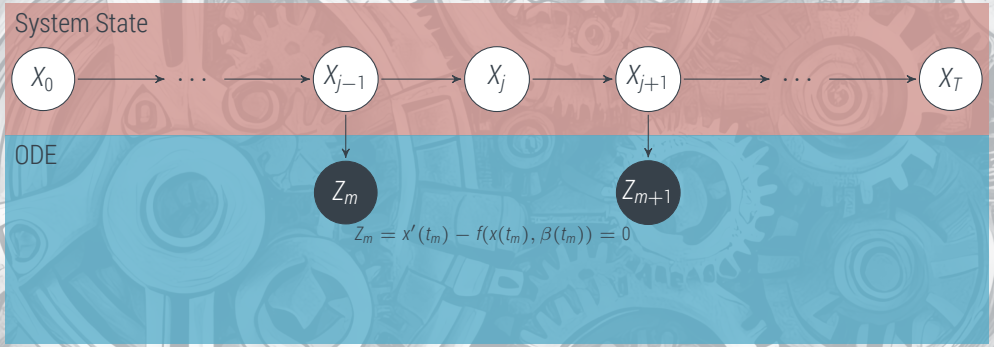
Simulation is inference from *information operators*

Bayesian filtering as a primitive for information assimilation



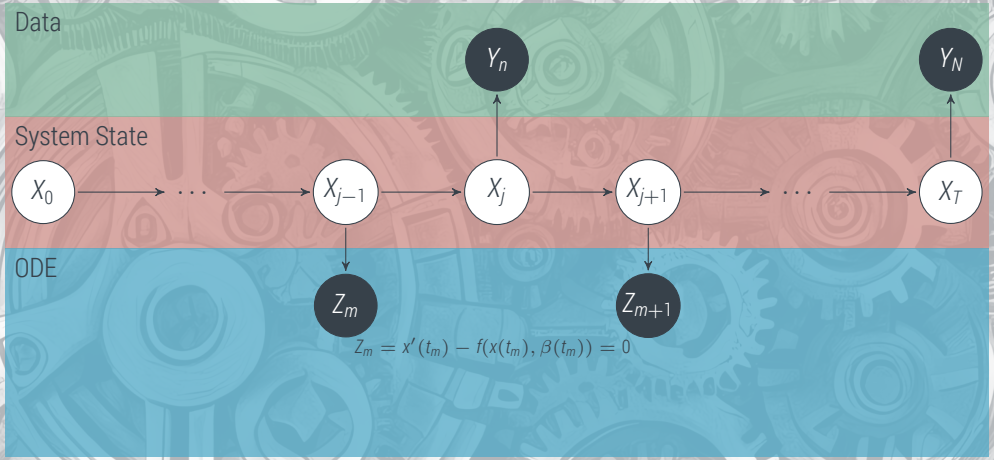
Simulation is inference from *information operators*

Bayesian filtering as a primitive for information assimilation



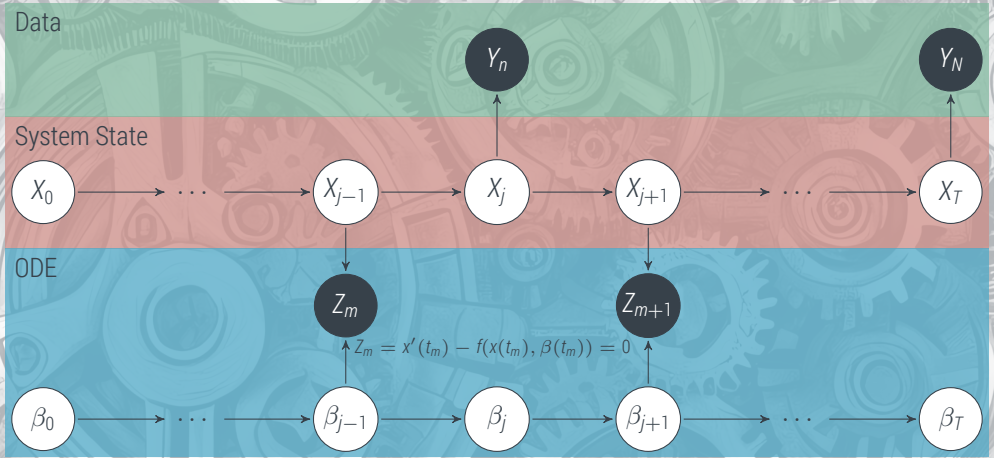
Simulation is inference from *information operators*

Bayesian filtering as a primitive for information assimilation



Simulation is inference from *information operators*

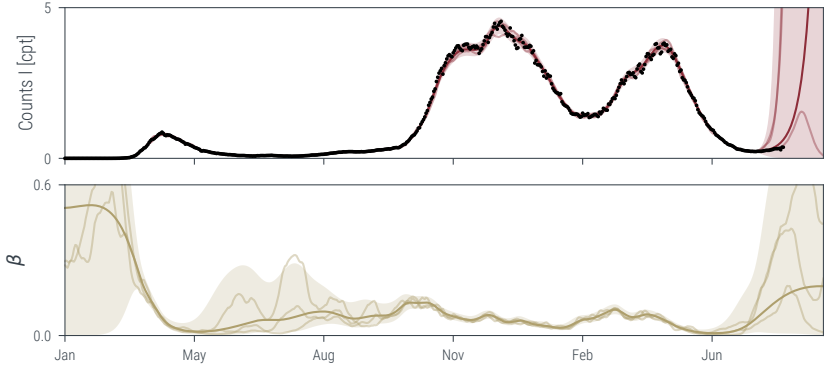
Bayesian filtering as a primitive for information assimilation



Simulation

Predict what will happen next, model dynamical systems

plot from Schmidt, Krämer, Hennig, NeurIPS 2021



Jonathan Schmidt

Nathanael Bosch

Simulation

Predict what will happen next, model dynamical systems

plot from Schmidt, Krämer, Hennig, NeurIPS 2021

In Lectures 4,5,6,7 we will see

- Ordinary differential equations can be solved by *Bayesian filters* operating on a Markov Chain
- Bayesian filtering is thus the generic language for simulation through time
- it can handle various forms of *information operators*, including measurements of the trajectory and conservation laws
- with structured state spaces, they can also solve PDEs and DAEs

Jonathan Schmidt

Nathanael Bosch

Integration

the central operation of Bayesian inference

Bayesian inference

$$p(x \mid y) = \frac{p(x, y)}{\int p(x, y)\, dx}$$

requires computing *integrals* (typically against probability measures)

$$F = \int f(x)\, dp(x) \quad \text{e.g.} \quad p(y) = \int p(y \mid x)\, dp(x)$$

In Lectures 8 & 9, we will see

- high-dimensional integration is very hard. It is typically done by Markov Chain Monte Carlo Methods, which are blunt tools. Nevertheless, knowing how they work can help build good special solutions for concrete settings.
- low-dimensional integrals, however, can be solved efficiently with *quadrature*.
- Classic quadrature methods can be thought of as a form of Gaussian process regression. Knowing how they work can make things even faster.

Philipp Hennig

Empirical Risk Minimization is arguably the most basic template of machine learning

$$\theta_* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \ell(y_i; \boldsymbol{\theta})$$

- This is an *optimization* problem (at least at first sight)
- It is the basic paradigm for **differentiable programming**, including deep learning, both supervised and unsupervised.
- With **automatic differentiation** we can compute

$$\nabla_\theta \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \ell(y_i; \boldsymbol{\theta})$$

- Methods making use of these gradients have been developed since the 50s (in *operations research* and *mathematical programming*)

But in practice, N is always **big**. So we compute a *batch estimate*

$$\nabla_\theta \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \ell(y_i; \boldsymbol{\theta}) \approx \frac{1}{M}\sum_{j=1}^{M} \nabla_\theta \ell(y_j; \boldsymbol{\theta}) =: \nabla_\theta \hat{\mathcal{L}}(\boldsymbol{\theta}) \qquad M \ll N$$

$$p(\nabla_\theta \hat{\mathcal{L}} \mid \nabla_\theta \mathcal{L}) \approx \mathcal{N}\left(\nabla_\theta \hat{\mathcal{L}}; \nabla_\theta \mathcal{L}, \Sigma \in \mathcal{O}\left(\frac{N-M}{NM}\right)\right)$$

- So there is a *likelihood* in our computation now! And it's not trivial!
- Where is that likelihood in your code?

```python
from torch.nn import CrossEntropyLoss, Linear

X, y = load_data()
model = Linear(784, 10)
lossfunc = CrossEntropyLoss()
loss = lossfunc(model(X), y)
loss.backward()

for param in model.parameters():
    print(param.grad)  # this is LHat. Where is Sigma?
```

Contemporary ML computations are fundamentally imprecise

Computing variance estimators is *not* expensive!

Dangel, Künstner, PH, ICLR 2020, `https://backpack.pt`

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \nabla \ell(y_i; \boldsymbol{\theta}) \quad \approx \frac{1}{M} \sum_{j=1}^{M} \nabla \ell(y_j; \boldsymbol{\theta}) \quad =: \nabla \hat{\mathcal{L}}(\boldsymbol{\theta})$$

already computed anyway!

$$\boldsymbol{\sigma}^2(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} [\nabla \ell(y_i; \boldsymbol{\theta})]^{\cdot 2} \approx \frac{1}{M} \sum_{j=1}^{M} \left[\nabla \ell(y_j; \boldsymbol{\theta})\right]^{\cdot 2} =: \hat{\boldsymbol{\sigma}}^2(\boldsymbol{\theta})$$

variance adds minimal overhead

```python
from backpack import extend, backpack, Variance

model = extend(Linear(784, 10))
lossfunc = extend(CrossEntropyLoss())
loss = lossfunc(model(X), y)

with backpack(Variance()):
    loss.backward()

for param in model.parameters():
    print(param.grad)  # this is LHat. Where is sigma2?
    print(param.variance)  # here it is!
```

Differentiable training needs a re-think

machine learning is *not* just optimization

In Lectures 10,11,12,13, we will see:

- deep training is essentially an unsolved problem
- uncertain computation plays a central role in this challeng
- many observables can be computed efficiently beyond the batch mean gradient
- these can be used
  - to monitor and control training, and
  - to add and calibrate Bayesian uncertainty to the trained network, at minimal overhead
- even then, many hyperparameters currently still have to be tuned in an outer loop. We will discuss how to do this efficiently

Frank Schneider

Lukas Tatzel

Agustinus Kristiadi

Julia Grosse

Notice something?

- A method is *numerical* if it makes *predictions* that can be *wrong*
- Linear Algebra methods become faster if we allow them just *estimate* the exact solution
- Simulation methods that are Bayesian filters, conditioned on various information sources
- Deep Training involves a *likelihood* within the computation itself

An Observation

Numerical Methods *are* elementary learning machines

A numerical method **infers** a **latent** (intractable) quantity z **from tractable observations** c (results of computations). This is exactly the task of machine learning, and formalised by **Bayesian inference**.

$$p(z \mid c) = \frac{p(c \mid z)p(z)}{\int p(c, z)\, dz}$$

It's just that the *data* isn't loaded from the **disk**, but directly collected by the **chip**. This doesn't change much, as long as we make sure to keep the cost of this data collection and integration low.

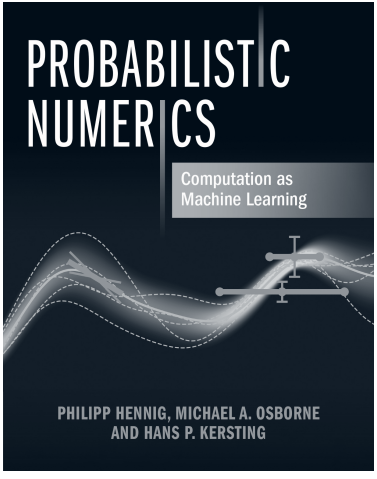
The tools we use to train the machines are also machines themselves!

And if you know how to build and run machines, you know how to build tools…

- **compute** is a source of information (numbers from the **chip**), just like empirical **data** (numbers from the **disk**)
- one can build **probabilistic numerical methods** that

$$p(z \mid c) = \frac{p(c \mid z)p(z)}{\int p(c, z)\, dz}$$

  1. formulate **prior** distributions over their target quantity z
  2. combine them with **likelihood** terms relating to the computed quantities c
- when computation is phrased as learning this way
  1. *empirical* and *computational information* can be combined more flexibly
  2. *uncertainty* from *finite and imprecise computation* can be tracked explicitly
  3. *algorithmic hyperparameters* may be *inferred* automatically



Why should you take this class?

Become an expert operator

- This is *not* a numerical analysis class! Proofs and theorems will only show up in passing.
- You will write *code* in this course, for credit (see below)
- This *is* a machine learning class

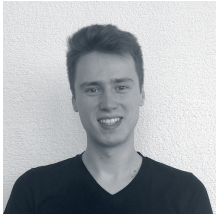
Probabilistic numerics is emerging as a holistic mathematical formalism for **data-centric computation**. Thinking in terms of inference will

- allow you to understand what happens inside of the closed box
- empower you to build better solutions for contemporary, composite machine learning tasks. ML solutions that "play nicely" with their surroundings and save resources
- Algorithmic knowledge is an extremely valuable expertise, sought after in the most advanced ML companies and in academia

The Team

A Master class, taught by experts with hands-on, detailed experience

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN



Marvin Pförtner
Linear Algebra, PDEs

Jonathan Wenger
Linear Algebra

Nathanael Bosch
ODEs

Jonathan Schmidt
Filtering, MCMC

Lukas Tatzel
Deep Training

Frank Schneider
Deep Training

Agustinus Kristiadi
Uncertainty for DL

Julia Grosse
Hyperparameter Tuning

– Admin –

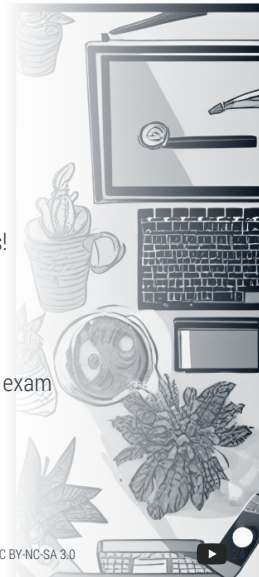
Exercises, and Points

How to be admitted to the exam

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

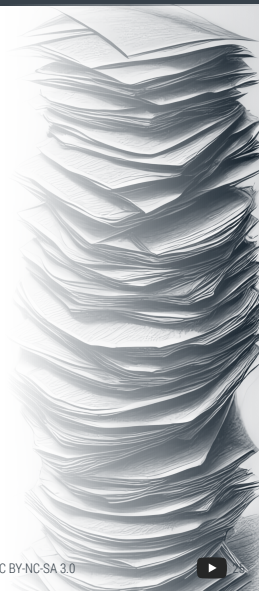
- Every content lecture comes with an **Exercise Sheet**, and an associated **Tutorial**

  Tutorials take place **Fridays, 12:15–14:00**, MvL6 Lecture hall
- Sheet #1 of 13 will be distributed *next week*, first tutorial on **4 November**
- Each sheet contains
  1. An **EXAMple** question, which gives an intuition for the exam. No need to submit solutions!
  2. A **extended coding exercise** in a `jupyter` notebook
- you can submit in groups of ≤ 2 persons
- Exercises are graded with a binary score ("sufficient")
  - You need at least 10 *sufficients* (i.e. 5 whole sheets, or 10 exercises) to be admitted to the exam
  - **A complete ("sufficient") exercise sheet amounts to two (2) bonus points in the exam**.
    (The exam has 100 points in total. $13 \cdot 2 = 26\%$ of the total score)
- Submission process: Export notebook **as a pdf** (with outputs), upload on Ilias.



The Exam

the annoying part



The exams are scheduled as follows:

1st Exam on **Monday, 13 February 2023, 08:00–10:00**

2nd Exam on **Friday, 31 March 2023, 08:00–10:00**

- You do **not** have to participate in the first exam to take the second. But if you fail the second exam, there will be no immediate opportunity for a re-sit.
- The EXAMples on each sheet provide an idea of what to expect in the exam.

Recordings

see camera in the back

- We are recording each lecture. Recordings will be put on youtube at

  `https://www.youtube.com/c/TübingenML`

- Recordings will *not* be available directly after the lecture, but only towards the end of term.
- By attending the lecture you are agreeing to potentially be recorded (we are not actively trying to record the audience, but wide-angle shots may include you, and questions may be audible).

MSc Theses & PhD Positions available!

Get a detailed intro to our work in this course

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

We're looking for several new Team members starting in 2023, to work on the following topics:

- probabilistic simulation of climate and weather
- tight integration of simulation and deep learning
- novel software engineering tools for deep learning
- and more…

Summary

- numerical algorithms are the *engines* of Machine Learning
- numerical methods also *are* learning machines
- this course will explain how they work, in the language of inference, so you can build ML solutions that run **faster**, **more reliably**, and are **easier to use**

Please cite this course, as

```
@techreport{NoML22,
    title = {Numerics of Machine Learning},
    author = {N. Bosch and J. Grosse
    and P. Hennig and A. Kristiadi
    and M. Pförtner and J. Schmidt
    and F. Schneider and L. Tatzel
    and J. Wenger},
    series = {Lecture Notes in Machine Learning},
    year = {2022},
    institution = {Tübingen AI Center},
}
```

Join us for the world's first dedicated Master Class on Algorithms *for* ML.