Bachelorarbeit Kognitionswissenschaft

Neural ODEs for latent force inference in dynamical systems

Felix Böhm

02. Mai 2023

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Kognitionswissenschaft

Gutachter

Prof. Dr. Philipp Hennig
Universität Tübingen
Methoden des Maschinellen Lernens

Betreuer

Nathanael Bosch
Universität Tübingen
Methoden des Maschinellen Lernens

Bearbeitungszeitraum: 17.11.2022-02.05.2023

Abstract

Differential equations, and in particular, ordinary differential equations (ODEs), are a central aspect of scientific machine learning. For latent force inference in models based upon them, traditional iterative optimization is expensive. Recently, new methods have emerged from the area of probabilistic numerics, which can perform inference while only solving the ODE once, yielding a posterior distribution over the inferred latent force. We repeat one of their experiments, in which they inferred the contact rate of a SIRD model on data of the COVID-19 outbreak, and show that just an inferred contact rate alone can not properly explain the data. Then, we try to approximate a posterior distribution using Laplace approximations, in order to quantify uncertainty.

Contents

Chapter 1

Introduction

Differential equations play a central role throughout many different fields of science, such as physics, chemistry, medicine, meteorology, sociology or epidemiology, to give only a few examples, where they very often are a core component of all kinds of different models and mechanisms. It is thus no surprise that the area of *scientific machine learning* (SciML) [BAB+19], combining scientific modeling, prominently via differential equations, with machine learning methods, has seen numerous recent developments and advances.

For example, [RPK17] and [RPK19] introduced *physics informed neural networks* that can encode prior mechanistic knowledge in the form of *partial differential equations* (PDEs). This encoded prior knowledge can vastly reduce the amount of data required for training the neural network, which is in cases where data is not abundant and expensive to procure, such as physics, a requirement that most other modern machine learning methods (in particular deep learning) can not fulfill.

Other interesting advancements have been made eg. by [CRBD19], who introduced the concept of *Neural ODEs*. These are essentially differential equations modeling residual networks, where the hidden state is modeled by a neural network. The resulting model is similar to a residual or recurrent neural network, but is continuous in terms of layer depth or time respectively. It additionally provides the possibility to choose different ODE solvers (introduced in Section 2.1.3) and tolerances in order to achieve a tradeoff between accuracy and computation time. Furthermore, they emphasized that it is possible to backpropagate through any ODE solver using adjoint and sensitivity analysis in constant memory and without the need to backpropagate through individual solver calculations, making training of larger models feasible.

Another big contribution was lately made by [RMM+21]. They published the *SciML software ecosystem*, which provides a big suite of high-performant tools for scientific modeling. Sharing a common mathematical foundation they call *universal differential equations* (UDEs), all individual parts of the SciML ecosystem together enable very efficient and simple to use tools for scientific modeling and learning those models via other machine learning methods, such as gradient-based optimization.

However, while learning differential equation based models or parameters thereof via iterative optimization methods is very powerful, it is also very expensive. Typically, with these methods, the differential equation has to be solved in every iteration, which increases computational

requirements and computation time for training such models, in particular with very large and complex models. To address this problem, [SKH21] have very recently developed a new algorithm that allows inference on latent forces in ODEs while only needing to run a solver once. Their approach, built on the foundation of *probabilistic numerics* [HOK22], employs Bayesian filtering to solve ODEs [TKSH19], yielding both a posterior over the ODE solution, as well as the latent force.

In light of this new approach, this thesis aims to replicate the experiment by [SKH21] in which they inferred the contact rate of an epidemiological SIRD model (see Section 2.1 for an introduction), but using a more traditional approach based on iterative optimization of point estimates. We will also infer the contact rate similarly to [SKH21], at first for a generated data set with known true contact rate, and then on a real data set. Additionally, we will extend the model to also infer the recovery and death rate, and then try to quantify uncertainty using Laplace approximations (see Section 2.5.1). The overall goal of the thesis is to obtain a baseline for this model and data set, with which the newer approaches can be compared.

Chapter 2 contains the theoretical foundations of this thesis, Chapter 3 details the method and implementation. Chapter 4 contains the individual experiments conducted and discusses their results, and finally, Chapter 5 provides a summary and conclusion of the thesis.

Chapter 2

Foundations

2.1 The susceptible, infected, recovered and deceased model

2.1.1 Susceptible, infected and recovered-type models

In epidemiology, susceptible, infected and recovered (SIR)-type models are a simple tool for modeling epidemics [Het00]. They describe transitions of individuals between different classes through a system of *ordinary differential equations* (ODEs), and therefore constitute, given initial values for every class, an ODE initial value problem.

SIR-type models assume a population of constant size P, which is subdivided into at least the class S for individuals susceptible to the disease, the class I for infected individuals, and the class R for individuals who recovered from the disease. However, many models of the SIR-type family extend the model, for example by also modeling changes in the population size, making it more suitable for longer term (endemic) modeling [Het00] or by introducing more classes, such as for example the SIDARTHE model proposed by [GBB$^+$20] that they used to model the COVID-19 epidemic outbreak in Italy, or the SEIR model that was used by [LZG$^+$20] to model the COVID-19 epidemic outbreak in Wuhan, including individual and governmental reactions.

This thesis will use the SIRD model, also being an extension of the classical SIR model, to fit and infer parameters of the COVID-19 epidemic outbreak in Germany. The next section will describe that model. The SIRD model, which is used in this project, introduces the class D which represents individuals that died due to the disease.

2.1.2 Mathematical formulation of the SIRD model

The susceptible, infected, recovered and dead (SIRD) model [Het00] extends the standard S, I and R classes (see Section 2.1.1) by a fourth class D, representing deceased individuals. The model is an initial value problem (IVP), and models the transitions between these classes using the ODE system

$$\begin{aligned} \dot{S}(t) &= -\beta S(t)I(t)/P & \dot{R}(t) &= \gamma I(t) \\ \dot{I}(t) &= \beta S(t)I(t)/P - \gamma I(t) - \eta I(t) & \dot{D}(t) &= \eta I(t) \end{aligned} \tag{2.1}$$

with initial values $S(0) = S_0\,, I(0) = I_0\,, R(0) = R_0, D(0) = D_0$.

In these equations, the variable $\beta \in [0\,, 1]$ represents the contact rate between individuals, $\gamma \in [0\,, 1]$ the recovery rate and $\eta \in [0\,, 1]$ the mortality rate. Later on, in Section 4.3.1, we will also extend the model by replacing these constants with time-dependent functions, just like β.

Since the SIRD model is an ODE initial value problem with no closed form solution, we need to numerically estimate the individual values for $S(t)\,, I(t)\,, R(t)\,, D(t)$ over time. This numerical approximation can be computed by ODE solvers, which will be introduced in the next section.

2.1.3 Numerical solvers for ordinary differential equations

Numerical solvers for ordinary differential equations (ODE solvers) can, as the name implies, approximately solve ODE initial value problems by computing a numerical approximation at discrete time steps. There are many different ways of obtaining such solutions, for example the very basic Euler's method [Eul68]: Given an ODE $\dot{y}(t) = f(t, y(t))$ with initial value $y(t_0) = y_0$, a step size $h > 0$, and discrete time steps $t_{n+1} = t_n + h$, Euler's method now approximates $y(t)$ by simply taking steps of size h along the derivative, resulting in the approximation

$$y(t_{n+1}) = y(t_n) + h\dot{y}(t_n) = y(t_n) + hf(t_n, y(t_n))\,. \tag{2.2}$$

In this equation, we can see that the left-hand side, which is the approximation at the next time step $y(t_{n+1})$, is explicitly defined in terms of f, the step size h and the approximation of the previous time step $y(t_n)$. which makes Euler's method an *explicit* ODE solving method. There is a multitude of different explicit ODE solvers, such as for example the class of (explicit) Runge-Kutta methods (see for example [But08]), which also contains Euler's method.

Some ODEs are *stiff*, which means that many solvers compute unstable solutions with normal step sizes. For stiff ODEs, it is usually better to use implicit methods. In implicit methods, the above condition does not hold: The left-hand term $y(t_{n+1})$ is only implicitly defined, which means it also occurs on the right-hand side, and, because of that, additional non-linear equations need to be solved. For example, there is an implicit version of Euler's method (as explained in [But08]), which looks almost the same as the explicit Euler method, except that on the right-hand side, $y(t_n)$ is replaced by $y(t_{n+1})$, resulting in

$$y(t_{n+1}) = y(t_n) + h\dot{y}(t_n) = y(t_n) + hf(t_n, y(t_{n+1}))\,, \tag{2.3}$$

where we then need to solve this new algebraic equation for $y(t_{n+1})$. This means that implicit methods are generally more expensive to compute (per step), but they are usually more stable in the case of stiff ODEs. Prominent are, again, the class of implicit Runge-Kutta solvers, which for example also contains the implicit version of Euler's method.

In this thesis, we will use a solver belonging to the class of Rosenbrock methods [PTVF07], which can be seen as semi-implicit: They work similarly to the aforementioned implicit methods, but instead of fully solving the non-linear equations at each step, they only perform one step toward that solution, and are also suitable for stiff problems [But08].

Implementations of ODE solvers usually allow for automatic step size control [But08]: They allow specification of an error tolerance, and can choose the next step size automatically at every iteration.

2.2 Maximum likelihood estimation

Given some data set $\mathcal{D} = \{(t_1, d_1), \dots, (t_n, d_n)\}$ and a model distribution $\mathcal{M}$, we can formulate the *likelihood* of the data $\mathcal{D}$ given $\mathcal{M}$, which is the probability of the data $\mathcal{D}$ occurring under $\mathcal{M}$:

$$\mathcal{L}(\mathcal{M}\,;\mathcal{D}) = p(\mathcal{D} \mid \mathcal{M}) = p((t_1, d_1) \mid \mathcal{M}) \cdots p((t_n, d_n) \mid \mathcal{M}) = \mathcal{M}(t_1) \cdots \mathcal{M}(t_n)\,. \tag{2.4}$$

The principle of maximum likelihood estimation (MLE) now is to choose the model distribution $\mathcal{M}$ such that the likelihood $\mathcal{L}(\mathcal{M}\,;\mathcal{D})$ is maximized, or, in other words, to choose the model under which the data has the highest probability.

However, in practice, we usually can't just estimate an arbitrary model distribution $\mathcal{M}_\theta(t)$. For obtaining MLE estimates, usually a parametric distribution is chosen for $\mathcal{M}_\theta(t)$, and instead of finding a distribution that maximizes the data, we only need to maximize over θ, leading us to the MLE estimate

$$\hat{\theta}_{\mathrm{ML}} = \operatorname*{argmax}_\theta \mathcal{L}(\theta\,;\mathcal{D})\,. \tag{2.5}$$

2.2.1 Normal distribution

If we assume that the model distribution is a (multivariate) normal distribution with mean vector $f_{\boldsymbol{\theta}}(t) \in \mathbb{R}^k$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{k\times k}$, and thus have a model distribution

$$p(\boldsymbol{d}_i) = \mathcal{N}(\boldsymbol{d}_i\,; f_{\boldsymbol{\theta}}(t_i)\,, \boldsymbol{\Sigma})\,, \tag{2.6}$$

where we want to calculate the MLE estimate for $\boldsymbol{\theta}$ (but not for $\boldsymbol{\Sigma}$), we can rearrange the MLE from Equation 2.5 into

$$\begin{aligned}
\hat{\boldsymbol{\theta}}_{\mathrm{ML}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}\,;\mathcal{D}) && (2.7)\\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} p(\mathcal{D} \mid \boldsymbol{\theta}) && (2.8)\\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p(\boldsymbol{d}_i \mid f_{\boldsymbol{\theta}}(t_i)) && (2.9)\\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log \mathcal{N}(\boldsymbol{d}_i\,; f_{\boldsymbol{\theta}}(t_i)\,, \boldsymbol{\Sigma}) && (2.10)\\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} -\frac{1}{2} \sum_{i=1}^{n} k \log 2\pi + \log\det(\boldsymbol{\Sigma}) + (\boldsymbol{d}_i - f_{\boldsymbol{\theta}}(t_i))^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{d}_i - f_{\boldsymbol{\theta}}(t_i)) && (2.11)\\
&= \operatorname*{argmax}_{\boldsymbol{\theta}} -\frac{1}{2} \sum_{i=1}^{n} (\boldsymbol{d}_i - f_{\boldsymbol{\theta}}(t_i))^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{d}_i - f_{\boldsymbol{\theta}}(t_i)) && (2.12)\\
&= \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{2} \sum_{i=1}^{n} (\boldsymbol{d}_i - f_{\boldsymbol{\theta}}(t_i))^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{d}_i - f_{\boldsymbol{\theta}}(t_i))\,, && (2.13)
\end{aligned}$$

which, if $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1\,, \dots\,, \sigma_k)$, can be further simplified to

$$\hat{\boldsymbol{\theta}}_{\mathrm{ML}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{k} \sigma_k^{-1} (\boldsymbol{d}_{ik} - f_{\boldsymbol{\theta}}(t_i)_k)^2\,, \tag{2.14}$$

and, if $\boldsymbol{\Sigma} = \boldsymbol{I}_k$, even further to obtain

$$\hat{\boldsymbol{\theta}}_{\mathrm{ML}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \|\boldsymbol{d}_i - f_{\boldsymbol{\theta}}(t_i)\|_2^2 \,. \tag{2.15}$$

Therefore, maximizing the likelihood under a normal distribution is equal to minimizing the L_2-loss if the covariance matrix is the identity matrix. If the covariance matrix is diagonal, but not the identity matrix, the individual entries σ_i correspond to different weighting of their respective dimension, larger values will lead to lower weighting.

2.2.2 Lognormal distribution

The (multivariate) lognormal distribution describes the distribution of a random variable whose (element-wise) logarithm is normally distributed:

$$\boldsymbol{d} \sim \mathrm{Lognormal}\left(\log \boldsymbol{\theta}\,, \boldsymbol{\Sigma}\right) \Leftrightarrow \log \boldsymbol{d} \sim \mathcal{N}(\log \boldsymbol{\theta}\,, \boldsymbol{\Sigma}) \tag{2.16}$$

The lognormal distribution is very useful for describing random variables that are assumed to be influenced by various independent factors, but the effects are multiplicative as opposed to additive as in the regular Normal distribution [LSA01], and are frequently used throughout different disciplines of science, for example in medicine ([Sar50], [Sar52], [Sar66], [Kon77] as cited by [LSA01]), economics ([CG05]) and many more.

Since we have defined the lognormal distribution in terms of the normal distribution, we can derive the MLE estimate (if $\boldsymbol{\Sigma}$ is diagonal) from Equation 2.14 under the same preconditions, except that the model distribution is a lognormal distribution, by simply replacing $\boldsymbol{d}_i$ with $\log \boldsymbol{d}_i$, and $f_{\boldsymbol{\theta}}(t_i)$ with $\log f_{\boldsymbol{\theta}}(t_i)$, and obtain

$$\hat{\boldsymbol{\theta}}_{\mathrm{ML}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{k} \sigma_k^{-1} (\log \boldsymbol{d}_{ik} - \log f_{\boldsymbol{\theta}}(t_i)_k)^2 \,. \tag{2.17}$$

Similarly from Equation 2.15 and if $\boldsymbol{\Sigma} = \boldsymbol{I}$, it follows that

$$\hat{\boldsymbol{\theta}}_{\mathrm{ML}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \|\log \boldsymbol{d}_i - \log f_{\boldsymbol{\theta}}(t_i)\|_2^2 \,. \tag{2.18}$$

Therefore, we can see that in the case of the lognormal distribution, $\hat{\boldsymbol{\theta}}_{\mathrm{ML}}$ minimizes the L_2-Distance between the log-transformed data points and the logarithm of the mean. This loss objective will be used later for optimizing, and will be referred to as *sum squared log space error* (SSLSE). For numerical reasons (especially if the data set contains zeroes), it is also beneficial to add 1 before taking the logarithm in the loss, which would be equivalent to add 1 to every point in the data set (and the model prediction) and then taking the SSLSE of that. For individual points $\boldsymbol{y}, \hat{\boldsymbol{y}} \in \mathbb{R}^n$, the SLSE used in this thesis is therefore given by

$$\mathrm{SLSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \|\log(\boldsymbol{y} + \boldsymbol{1}) - \log(\hat{\boldsymbol{y}} + \boldsymbol{1})\|_2^2 \,, \tag{2.19}$$

where the logarithm is applied element-wise. The SSLSE is then the sum of the SLSE of the individual data-prediction pairs.

Also, note that here, we modeled the first parameter of the lognormal distribution with $\log f_{\boldsymbol{\theta}}$ instead of with $f_{\boldsymbol{\theta}}$ directly. By doing this, we obtain the equivalence

$$\boldsymbol{d}_i \sim \text{Lognormal}\left(\log f_{\boldsymbol{\theta}}(x_i)\,, \boldsymbol{\Sigma}\right) \Leftrightarrow \boldsymbol{d}_i = f_{\boldsymbol{\theta}}(x_i) * \epsilon\,, \epsilon \sim \text{Lognormal}\left(\mathbf{0}\,, \boldsymbol{\Sigma}\right)\,, \tag{2.20}$$

which means that $f_{\boldsymbol{\theta}}$ is still a point estimate for the data points, with a lognormally distributed multiplicative error, analogously to the additive error with the normal distribution. If we had instead modeled the parameter of the lognormal distribution with $f_{\boldsymbol{\theta}}$ directly, we would get point estimates for the data with $\exp(f_{\boldsymbol{\theta}})$.

2.3 Maximum a posteriori estimation

The MLE estimate is a point estimate of parameters that maximize the likelihood of given data, but it can not leverage any sort of knowledge or prior information about the paramateres that might exist already. However, if we do have prior information about the parameters in form of a prior distribution $p(\boldsymbol{\theta})$, we can also formulate the *maximum a posteriori* (MAP) estimate

$$\begin{aligned}
\hat{\boldsymbol{\theta}}_{\text{MAP}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta} \mid \mathcal{D}) && (2.21)\\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \frac{p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{D})} && (2.22)\\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) && (2.23)\\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \log \mathcal{L}(\boldsymbol{\theta}\,; \mathcal{D}) + \log p(\boldsymbol{\theta})\,, && (2.24)
\end{aligned}$$

which incorporates this prior knowledge. If we look again at the result from Section 2.2.2, and extend it by assuming a prior distribution $p(\boldsymbol{\theta}) \sim \mathcal{N}(\boldsymbol{\mu}^p\,, \boldsymbol{\Sigma}^p)$, where $\boldsymbol{\Sigma}^p = \text{diag}(\sigma_1^p\,, \dots\,, \sigma_k^p)$ is diagonal, then we can further rearrange Equation 2.24, analogously to Equation 2.14, resulting in

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \frac{1}{2} \sum_{i=1}^{n} \|\log \boldsymbol{d}_i - \log f_{\boldsymbol{\theta}}(t_i)\|_2^2 + \frac{1}{2} \sum_{j=1}^{k} (\sigma_k^p)^{-1} (\boldsymbol{\mu}_k^p - \boldsymbol{\theta}_k)^2\,. \tag{2.25}$$

This term, too, can be further simplified if the variances are equal ($\sigma^p = \sigma_1^p = \dots = \sigma_n^p$), to obtain

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \frac{1}{2} \sum_{i=1}^{n} \|\log \boldsymbol{d}_i - \log f_{\boldsymbol{\theta}}(t_i)\|_2^2 + \frac{(\sigma^p)^{-1}}{2} \|\boldsymbol{\mu}^p - \boldsymbol{\theta}\|_2^2\,. \tag{2.26}$$

Thus, assuming a Normal prior parameter distribution is equal to regularization with an L2 norm penalty of the parameters, keeping them close to the prior distribution parameter $\boldsymbol{\mu}^p$.

2.4 Optimization

In the previous sections, we have seen that maximizing the likelihood or the posterior is, in this case, equivalent to finding parameters that minimize some loss function. Very often, the minimizer of that loss function can not be calculated in closed form, and thus has to be numerically estimated. The process of finding such estimated minimizers is called *optimization*, and there are many different optimization algorithms that can accomplish this. This section introduces the three optimization algorithms that are used in this thesis.

2.4.1 Adam

The Adam optimization algorithm, introduced by [KB14], is a first-order optimization algorithm. It extends traditional *stochastic gradient descent* (SGD) by combining both the momentum method [Pol64] and RMSProp [TH+12]. In essence, it calculates per-parameter moving averages of the gradients (first momentum) and the (element-wise) squared gradients (second momentum) in order to provide momentum and a per-parameter adapted learning rate.

Adam works as follows: Let $g_k = \nabla_\theta f(\theta_{k-1})$ denote the gradient at iteration k. Then, the first momentum term is given by

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k \,, \hat{m}_k = \frac{m_k}{1-\beta_1^k} \,, \tag{2.27}$$

and the second momentum term by

$$v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k \odot g_k \,, \hat{v}_k = \frac{v_k}{1-\beta_2^k} \,. \tag{2.28}$$

The parameter updates are then given by

$$\theta_k = \theta_{k-1} - \lambda \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \epsilon} \,. \tag{2.29}$$

For some learning rate $\lambda > 0$ and $\epsilon > 0$. All other values are initialized as zero. Because they are zero initialized, m_k and v_k are biased (toward zero), $\hat{m}_k$ and $\hat{v}_k$ remove the bias.

2.4.2 Newton

Newton's method for optimization (as described in [NW99]) is a simple and fast converging second-order method. This means that, unlike first-order methods, it also makes use of second-order derivatives (the Hessian matrix) of the optimization objective. The Hessian and the first-order gradients are used in conjunction for a second-order Taylor approximation, a quadratic function, which is then minimized instead in every iteration. Let $H_k = \nabla_\theta \nabla_\theta f(\theta_{k-1})$ be the Hessian and $g_k = \nabla_\theta f(\theta_{k-1})$ be the gradient at iteration k. Then, Newton's method calculates the step direction as

$$\rho_k = -H_k^{-1} g_k \tag{2.30}$$

While taking the step ρ_k would minimize the Taylor-approximation in exactly one step, but in practice, only a small step is taken in that direction, or a line search algorithm (see Section 2.4.4) is used in order to determine an approximately optimal step length in that direction.

Newton's method, sounds compelling, but in practice, it's often not optimal due to two major problems: The first is that, while it can often converge fast, it often behaves in problematic ways when far from the minimum or in regions with many local minima. The other problem is that the seemingly harmless Equation 2.30 for the descent direction is often completely intractable to compute: It requires both the time-consuming calculation of the Hessian, which takes time $\mathcal{O}(n^2)$ (if n is the number of parameters), and on top of that, its inversion, which takes $\mathcal{O}(n^3)$, making it impossible to use in problems with many parameters.

2.4.3 BFGS algorithm

In order to reap the benefits of Newton's method, but also scale to bigger problems than Newton's method can address, one can instead resort to the family of *quasi-Newton* optimization algorithms. Algorithms of this family do not calculate the Hessian, but instead approximate it using only gradient information, and therefore stay first-order and way less computationally expensive.

Among the quasi-Newton optimization algorithms, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm (after [Bro70], [Fle70], [Gol70], and [Sha70]) is perhaps the most popular choice, and the one used in this thesis.

BFGS, as detailed in [NW99], works as follows: Let $\boldsymbol{\theta}_0$ be the starting value that should be optimized, and $f(\boldsymbol{\theta})$ the objective that should be minimized. Furthermore, let $\boldsymbol{B}_0 \approx \boldsymbol{H}_0^{-1}$ be the first (positive definite and symmetric) approximation of the inverse Hessian (which is often just set as the identity matrix, resulting in regular gradient descent in the first iteration, but other initial values are also possible).

Then, the k-th iteration of the BFGS algorithm goes as follows: First, obtain a direction p_k from the inverse Hessian approximation and the current gradient:

$$\boldsymbol{p}_k = -\boldsymbol{B}_k \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_k) \tag{2.31}$$

Then, calculate (approximately):

$$\alpha_k = \underset{\alpha}{\operatorname{argmin}} f(\boldsymbol{\theta}_k + \alpha_k \boldsymbol{p}_k) \tag{2.32}$$

In practice, α_k is found using a line search algorithm (see Section 2.4.4). This results in the increment $\boldsymbol{s}_k = \alpha_k \boldsymbol{p}_k$ and we can update $\boldsymbol{\theta}$ by this increment:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \boldsymbol{s}_k \tag{2.33}$$

The key part of BFGS is updating the approximation of the inverse Hessian. For this, set

$$\boldsymbol{y}_k = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_{k+1}) - \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_k) \tag{2.34}$$

and

$$\rho_k = \frac{1}{\boldsymbol{y}_k^\top \boldsymbol{s}_k} \tag{2.35}$$

The updated inverse Hessian approximation is then given by the formula

$$\boldsymbol{B}_{k+1} = (I - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^\top) \boldsymbol{B} (I - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^\top) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^\top \tag{2.36}$$

This procedure is repeated until a convergence criterion is reached. Both the computational and the time complexity of one iteration of the BFGS algorithm are quadratic in $|\boldsymbol{\theta}|$.

2.4.4 Line search

Both Newton's method (in practice) and BFGS perform line search in order to determine optimal step lengths. A line search algorithm, as described in [NW99], usually determines a direction $\boldsymbol{p}_k$ and a step length α_k in every iteration, which sufficiently minimize an objective function. In the case of the line search used in Newton's method and in BFGS, the direction

is already determined by the respective method, and only the optimal step length α_k needs to be determined, so a line search algorithm only needs to approximate

$$\alpha_k \approx \underset{\alpha}{\operatorname{argmin}} f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k)\,, \tag{2.37}$$

as for example required in Equation 2.32 for BFGS. There are various different line search algorithms, providing different balances between how good α_k is (ie. how good the function is minimized), and how many individual steps or evaluations of f are needed in order to determine it.

A simple line search method is for example backtracking (as written in [NW99]), which, as the name suggests, starts with an initial guess $\alpha^{(1)} = \overline{\alpha}$, and then successively decreases the current candidate exponentially by multiplying it with a factor $m \in (0, 1)$

$$\alpha^{(n+1)} = m\alpha^{(n)}\,, \tag{2.38}$$

until a sufficiently good $\alpha^{(n)}$ is found. In this case, $\alpha^{(n)}$ is deemed good if it satisfies

$$f(\boldsymbol{x}_k + \alpha^{(n)}\boldsymbol{p}_k) \leq f(\boldsymbol{x}_k) + c\alpha^{(n)}\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)^\top \boldsymbol{p}_k \tag{2.39}$$

for some chosen $c \in (0, 1)$.

Another more sophisticated line search algorithm based on conjugate gradients was described by [HZ06], which is what we will use in the majority of our experiments. We will refer to this line search algorithm as HagerZhang line search.

2.5 Uncertainty quantification

The MLE and MAP estimates introduced in Section 2.2 and Section 2.3 are simple point estimates. However, it is often very desirable to have a (posterior) distribution over the parameters instead, as this distribution directly reflects the uncertainty of the model regarding the parameters.

There have been different approaches toward this. For example, [Bis94] introduced *Mixture Density Networks*, which combines neural networks with mixture density models, and can also model other distribution parameters. A different possibility that has recently been explored for uncertainty quantification are ensemble methods [LPB17], which encompasses training multiple models at once and combining their individual predictions. Often used are also different kinds of Markov Chain Monte Carlo (MCMC) methods (see for example [BGJM11]), which can approximate probability densities by repeated sampling. MCMC methods can be very costly if good approximations are required, as many samples are required, and during sampling, many function evaluations are necessary, which can be especially problematic if the function is expensive, as is the case in the SIRD model, as every evaluation involves solving the ODE given by it.

Another very compelling way of uncertainty quantification that does not involve expensive sampling is realized through the usage of Laplace approximations, which are explained in the next section.

2.5.1 Laplace approximations

Laplace approximations were first introduced to the domain of machine learning by [Mac92], and recently, have been demonstrated to scale up to and be useful even in modern, bigger

networks [RBB18]. They provide an easy way of providing uncertainty estimation, and can be used post-hoc (after training on a point estimate) without having to change anything about the training itself [DKI+22]. Furthermore, they also enable marginal likelihood-based model selection and hyperparameter tuning [IBF+21].

The basic idea behind Laplace approximations is as follows: Equation 2.21 *almost* provides the formula of the posterior parameter distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$, but it is unnormalized, since the normalizing factor $p(\mathcal{D})$ (in Bayesian terms called the 'evidence' for the data $\mathcal{D}$) was dropped. This normalizing factor is intractable, and thus we can't calculate the true posterior.

This is where Laplace approximations come into play: They allow us to approximate this otherwise intractable distribution with a Gaussian: Let $\boldsymbol{\theta}^*$ be a mode of the unnormalized posterior $p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})$. The MAP estimate introduced in Section 2.3 suffices this per definition, so it is possible to obtain the MAP estimate first through regular optimisation, and then perform the Laplace approximation with $\boldsymbol{\theta}^* = \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}$. Then, we can approximate the posterior as

$$p(\boldsymbol{\theta} \mid \mathcal{D}) \approx \mathcal{N}(\boldsymbol{\theta}\,; \boldsymbol{\theta}^*, \boldsymbol{S}^{-1}) \tag{2.40}$$

Where $\boldsymbol{S}$ is the (positive definite) Hessian matrix of the log unnormalized posterior at the mode $\boldsymbol{\theta}^*$:

$$\boldsymbol{S} = -\nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \left(\log p(\mathcal{D} \mid \boldsymbol{\theta}^*) p(\boldsymbol{\theta}^*) \right) \tag{2.41}$$

As we are only working with a small number of parameters, completing the full Hessian matrix is completely tractable, and we do not have to approximate it through other means.

Since this approximation yields a distribution over the parameters $\boldsymbol{\theta}$ (given the data $\mathcal{D}$), we can use it to draw random weight samples from this distribution and parameterize the model with these samples, allowing us to obtain an (approximate) distribution of model outputs.

Chapter 3

General model and method

In this section, the data set and the model used in the experiments in Chapter 4 will be explained, as well as how the model will be optimized. The following will generally apply to all experiments, except where specified otherwise.

3.1 COVID-19 data set

As [SKH21], we will use the COVID-19 data set published by the Johns Hopkins University Center for Systems Science and Engineering [DDG20]. This data set contains, starting on January 22, 2020, the daily numbers of total confirmed COVID-19 cases (c_t), as well as the numbers of recovered (r_t) and deceased (p_t) individuals that died during their COVID-19 infection. The population size will be fixed to $P = 83.783.945$.

The complete training data set then is

$$\mathcal{D} = \{\boldsymbol{d}_0, \ldots, \boldsymbol{d}_T\}, \boldsymbol{d}_t = (S_t, I_t, R_t, D_t)^\top, \tag{3.1}$$

which can be obtained by transforming the aforementioned public data set:

$$\begin{aligned} S_t &= P - c_t \quad & I_t &= c_t - p_t - r_t \\ R_t &= r_t & D_t &= p_t \end{aligned} \tag{3.2}$$

We will, as [SKH21], only model the data until July 16, 2021, ie. $T = 541$, for a total of 542 days.

3.2 Model and implementation details

3.2.1 Parameterized SIRD model

In all experiments, the SIRD model introduced in Section 2.1.2 will be used to predict the number of susceptible (S), infected (I), recovered (r) and deceased (D) individuals of a given data set, by inferring the latent, time-varying parametric force β which represents the contact rate. The model itself will be denoted as

$$f_{\boldsymbol{\theta}}(t) = (S(t), I(t), R(t), D(t)), \tag{3.3}$$

where $S(t)$, $I(t)$, $R(t)$ and $D(t)$ are described by the ODE initial value problem formulated in Equation 2.1, except that β is now parameterized by $\boldsymbol{\theta}$:

$$\begin{aligned} \dot{S}(t) &= -\beta_{\boldsymbol{\theta}} S(t) I(t)/P & \dot{R}(t) &= \gamma I(t) \\ \dot{I}(t) &= \beta_{\boldsymbol{\theta}} S(t) I(t)/P - \gamma I(t) - \eta I(t) & \dot{D}(t) &= \eta I(t) \end{aligned}, \tag{3.4}$$

with given initial values $S(0) = S_0, I(0) = I_0, R(0) = R_0, D(0) = D_0$. The parameter $\boldsymbol{\theta}$ is omitted from S, I, R and D for better visual clarity.

3.2.2 Modeling the contact rate

In all experiments, the contract rate, represented by β, is to be inferred as a latent parametric force. It will be modeled as a weighted sum of N_β Gaussians with equal width σ_β and individual weight parameters θ_i, which will be the learnable parameters. On top of this, to ensure $\beta(t) \in [0, 1]$, a sigmoid function is applied to the sum. The individual Gaussians are each placed with their own extremum μ_i equidistantly across the interval $[0 - 2\sigma_\beta, T + 2\sigma_\beta]$, which means that the individual μ_i can be calculated as

$$\mu_i = \frac{i-1}{N_\beta - 1}(T + 4\sigma_\beta) - 2\sigma_\beta \,. \tag{3.5}$$

This *padding* by $2\sigma_\beta$ in either direction allows for equal prior weight distribution everywhere, even near the edges. Without padding, the prior weights for the Gaussian curves close to 0 or T would have to be scaled down. Figure 3.1b shows how the contact rate looks without padding.

The mathematical formula for the parametric contact rate model is then given by

$$\hat{\beta}(t) = \sigma\left(\sum_{i=1}^{N} \theta_i \phi_i(t)\right), \phi_i(t) = \exp\left(-\frac{(t-\mu_i)^2}{2\sigma_\beta^2}\right), \tag{3.6}$$

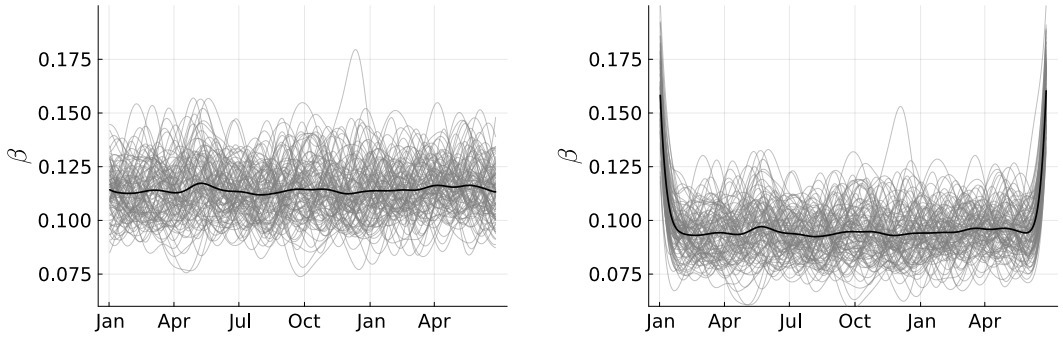
where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

In all experiments, $N_\beta = 36$ individual gaussian curves are used for the contact rate. We assume a prior Normal distribution of the weights, with mean $\mu_{w_\beta} = -1$ and standard deviation $\sigma_{w_\beta} = 0.01$. During initialization, the weights are independently and identically distributed (iid) drawn from this prior distribution. The resulting contact rates average at around 0.114 . Figure 3.1a shows prior samples of the contact rate from this prior distribution.

3.2.3 Other model parameters

For the first two experiments, in accordance to [SKH21], we are using a constant recovery rate of $\gamma = 0.06$ and a constant death rate of $\eta = 0.002$. In later experiments, the model will be extended and instead of assuming fixed constants for the recovery and death rate, they will also be modeled as a parametric latent force and inferred. The details of this extension can be found in Section 4.3.1.

The data sets used in the experiments are usually problematic for our SIRD model: They start with zero infections. Looking at the SIRD formulae (Equation 2.1 or Equation 3.4), it is immediately obvious that, if at any point in time $I(t) = 0$, then it follows that $I(t + \Delta t) = 0$ for all $\Delta t > 0$, or, in other words, if there are no infected individuals, nobody can catch an



(a) with padding (as used in the experiments)

(b) without padding

Figure 3.1: 100 Samples of the contact rate model with weights drawn from the prior distribution used in the experiments, as explained in Section 3.2.2. The grey lines show the individual samples, and the black lines show the average across the samples. Figure 3.1a shows the contact rate with padding as was used in the experiments, and Figure 3.1b shows, as can be seen by the higher contact rate at the beginning and end, why the padding is necessary.

infection. This makes perfect sense for the model, as it assumes a constant population and therefore does not model population size changes or people traveling in or out of the country, but it also means that there *have* to be infected individuals initially. We can capture these initially infected individuals in another (hyper-) parameter, which will be denoted as I_{initial}. Using this parameter, we can now use the initial values

$$S_0' = S_0 - I_{\text{initial}} \qquad I_0' = I_{\text{initial}} \qquad R_0' = R_0 \qquad D_0' = D_0\,, \tag{3.7}$$

if the problem $I_0 = 0$ arises. Note that, using this method, I_{initial} can also be learned in exactly the same manner as for example the weights for the modeled contact rate β. However, for our experiments, we will just set $I_{\text{initial}} = 1$, as we found no real benefit of also training this parameter.

For solving the SIRD equations, we use the 'Rosenbrock23' ODE solver (see Section 3.3.1), and we use a tolerance of 10^{-5} for both relative and absolute tolerances. This choice is explained in Appendix A.1.

3.3 Learning the parameters

The model parameters will be learned by minimizing the SSLSE (see Equation 2.19) between the data and the predicted I, R and D values. Like [SKH21], we will not use the class of susceptible individuals S for loss calculation, as we do not have real data for S, but just calculate it from the other classes based on the assumption of a constant population size.

For minimizing the loss, we will mainly make use of the BFGS algorithm introduced in Section 2.4.3. The initial approximation of the inverse Hessian in the first BFGS iteration is set to $\frac{0.1}{\|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})\|}\boldsymbol{I}$.

One possible pitfall with BFGS (and equally Newton) is that, during the line search, it is possible that parameters have to be evaluated for which the ODE given by the SIRD

equations becomes very stiff, to the point that the ODE solver we use produces unstable solutions. We circumvent this problem by returning a very large number (10^{10}) in the loss function if the ODE solution is unstable. Importantly, this approach also causes all gradient information to get lost, meaning that if an optimizer can only try parameters producing unstable solutions, no further optimization can occur. This is especially important during initialization, where this possible scenario has to be avoided by choosing sensible initializations (which the initialization described earlier suffices).

3.3.1 Implementation details

This thesis makes use of the SciML software ecosystem [RMM+21] for the Julia programming language [BEKS17]. All gradients are computed using *forward mode differentiation*, as implemented by [RLP16]. For optimization algorithms (as listed in Section 2.4), [MR18] is used, the ODE implementation and ODE solvers are provided by [RN17], and probability distributions are used as provided by [LWB+19]. All plots and graphics are made using [CSR+23]. The source code used for all experiments contained in this thesis is made available in a git repo[1].

All experiments are executed on a standard desktop grade computer with an AMD Ryzen 3600 CPU.

[1] `https://github.com/17ex/sird-contact-rate`

Chapter 4

Experiments

4.1 Inferring the contact rate of artificial data with known contact rate

In the first experiment, we will generate an artificial data set with a known contact rate, and evaluate how well the contact rate can be recovered. The goal is to show that, given all model assumptions are fulfilled, the true contact rate can be recovered.

4.1.1 Generating an artificial data set

In this experiment, the same model will be used as described in Section 3.2. However, instead of using real COVID-19 case numbers, instead, we generate an artificial data set, of which we know the true contact rate.

Here, we accomplish this by drawing the 'real' weights from some distribution, which determine the real contact rate. The distribution for this is a normal distribution, with mean $\mu_{\beta_1} = -0.78$ for the first 10 weights, $\mu_{\beta_2} = -1.275$ for the next 15 weights, and $\mu_{\beta_3} = -0.6$ for the last 11 weights. For all weights, the standard deviation is $\sigma_\beta = 0.01$. The numbers here are chosen arbitrarily, but with the intention of generating an interesting data set that can be interpreted: Using these numbers, we obtain a contact rate that starts out moderately high, then rapidly decreases and stays low for a while, and is high at the end.

Next, the true case numbers for S, I, R and D can be obtained by simply solving the SIRD equations, using the chosen parameters for the contact rate. On top of that, we will apply multiplicative noise, as we assume a lognormal model distribution. The noise factors ϵ_i are iid drawn from a lognormal distribution with $\mu_\epsilon = 0$ location and scale factor $\sigma_\epsilon = 0.01$. The generated data set, as well as the true contact rate, are depicted in Figure 4.1.

4.1.2 Method

The model used in this experiment is as described in Section 3.2, that is, we infer the contact rate, but assume a fixed recovery and death rate. MLE estimates for the parameters are determined using BFGS as explained in Section 2.4.3, with the HagerZhang line search (see Section 2.4.4).

4.1.3 Results

The final training loss (SSLSE) was 1.64228 (rounded), after 397 BFGS iterations in 9.5 minutes. Figure 4.1 shows the predicted S, I, R and D values, as well as the inferred contact rate. We can see that the model was able to recover the true contact rate well, with some slight deviations in the months around July 2020 and in the end (after April 2021). Both time spans with the deviations match up with the time spans of higher noise in the infected class I (or, in other words, with time spans of high infection counts, as the error is multiplicative).

4.1.4 Discussion

This experiment shows, using a toy example, that, at least if all the model assumptions are fulfilled, the model is able to recover the true contact rate reasonably well. Furthermore, this is even true if the prior contact rate guess is not close to the real one, and the data is moderately noisy.

However, while it worked well under these artificial conditions, we don't know how well this transfers to over to the real COVID-19 statistics. Particularly, we don't know how well the SIRD model with variable contact rate can explain them, or how well our model of the contact rate would work. In the next experiment, we will address these questions by applying the same model to the real data set, and see how well we can infer the contact rate on the real data set.

4.2 Inferring the contact rate of real COVID-19 data

This experiment examines whether we can also infer the contact rate of the German COVID-19 statistics using the same model that worked for the toy example in the previous experiment.

4.2.1 Method

We use both data set as described in 3.1 and model as described in Section 3.2, and try to infer the contact rate β, while assuming a constant recovery and death rate (see Section 3.2.3).

In this experiment, for obtaining the MLE estimates, will try out the backtracking line search (explained in Section 2.4.4, and with quadratic interpolation for the step size guess) instead of the HagerZhang line search used in the other experiments, to show the impact on performance different optimizers can have.

4.2.2 Results

The optimizer stops after 226 iterations, with a final error of 185.19889 (rounded). Using the backtracking line search algorithm, this takes only 9 seconds, which is approximately 60 times faster than the previous experiment. The resulting predictions of S, I, R and D, as well as the contact rate, are depicted in Figure 4.2.

However, the overall fit of the model predictions to the data is bad. Especially the predicted numbers of recovered R and deceased D individuals are way off, and the predictions for infected I are also inaccurate, especially during the first wave of infections at around April 2020, and between October 2020 and January 2021, around the second wave of infections.

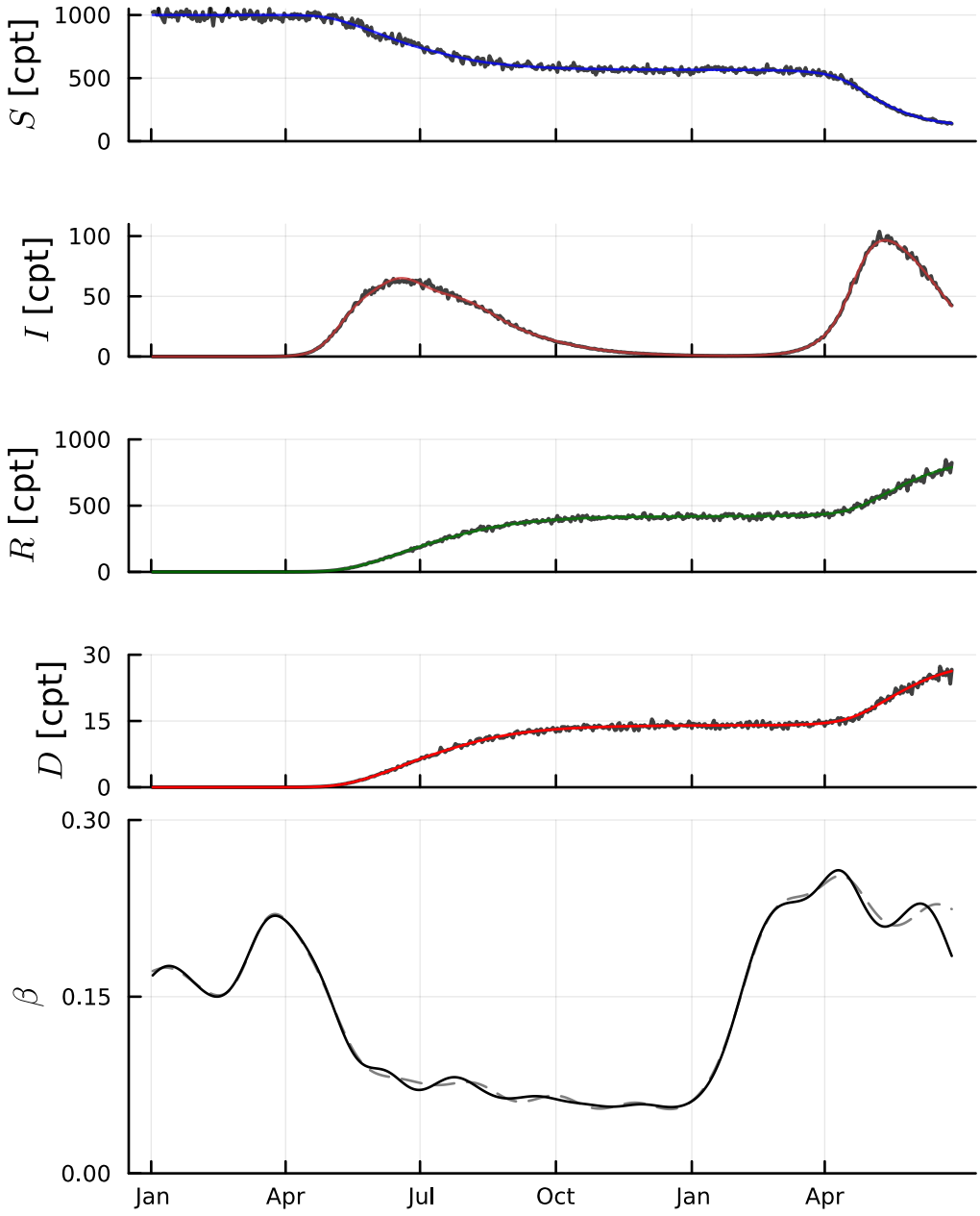


Figure 4.1: **Results of the first experiment**: The first four plots show the generated data set in black, and the predicted model values for S, I, R and D in color. All counts for S, I, R and D have been rescaled to cases per thousand (cpt). The last plot shows, in a dashed grey line, the chosen true contact rate, and in black the inferred contact rate.

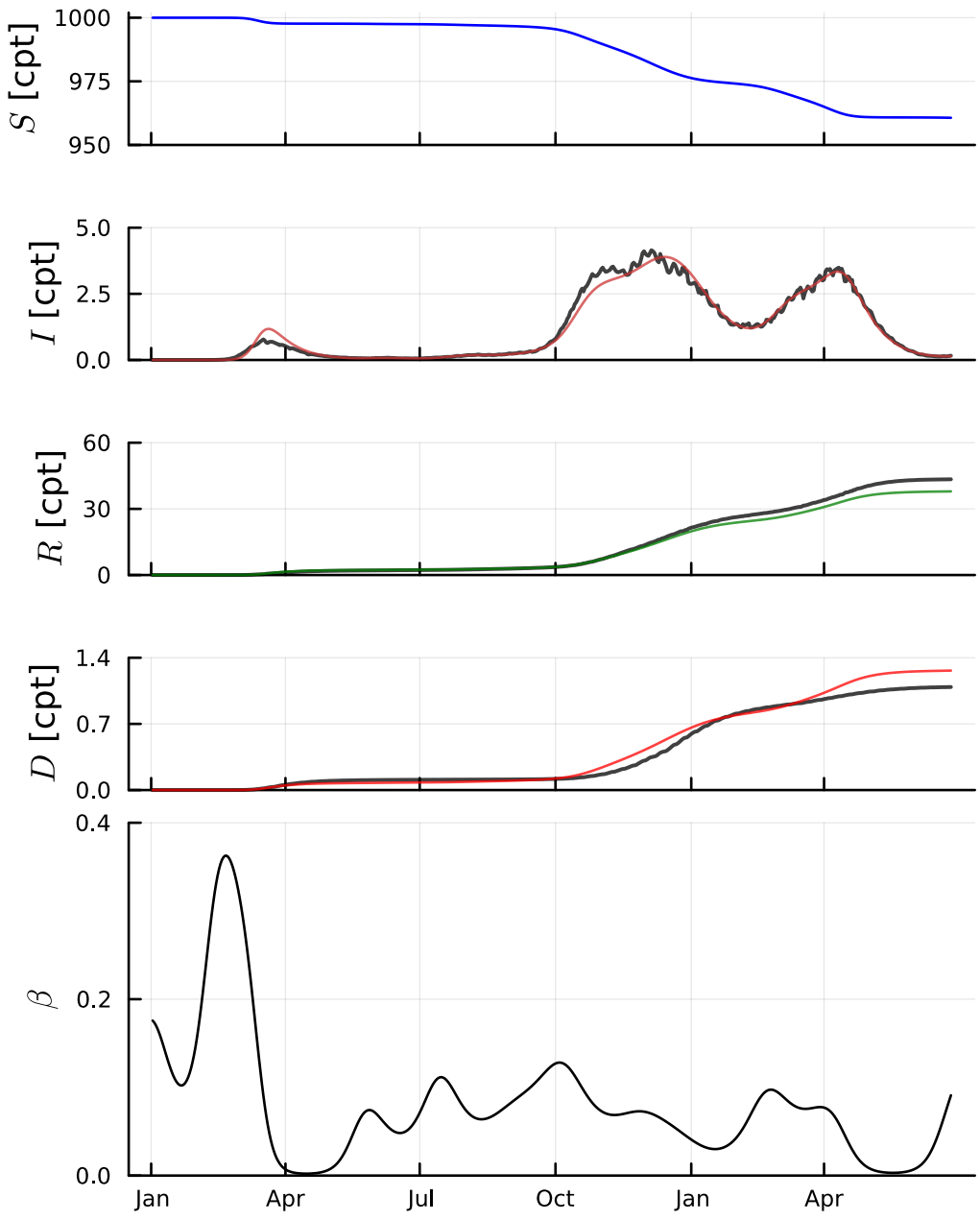


Figure 4.2: **Results of the second experiment**: Shown in the bottom plot is the MLE estimate for the contact rate β, the top four plots show in color the model predictions for S, I, R and D (rescaled to cases per thousand (cpt)), as well as the observed German COVID-19 statistics in black, for which the contact rate was inferred. As can be seen, the fit for counts of both recovered R and deceased D individuals is bad, as well as the counts of infected I individuals at around April 2020 and between October 2020 and January 2021.

4.2.3 Discussion

This experiment has shown that by only inferring the contact rate, the model can not accurately fit the observed data.

While a small contribution to the bad fit may be made by the choice of the backtracking line search for BFGS, and thus not obtaining a good enough solution, this experiment brings to light other issues: The model simultaneously underestimates the number of recovered R, and overestimates the number of deceased D. Since both only depend on I, we can therefore conclude that the assumed constant for the recovery rate γ and death rate η are not a good choice for the model, as the predictions can't match the data.

Going even further, if we take a look again at the SIRD equations (see Equation 2.1), and notice that

$$\begin{aligned} D(t) &= D_0 + \int_0^t \dot{D}(r)dr && (4.1)\\ &= D_0 + \int_0^t \eta I(r)dr && (4.2)\\ &= D_0 + \eta \int_0^t I(r)dr && (4.3)\\ &= D_0 + \frac{\eta}{\gamma} \int_0^t \gamma I(r)dr && (4.4)\\ &= R_0 - (R_0 - D_0) + \frac{\eta}{\gamma} \int_0^t \dot{R}(r)dr && (4.5)\\ &= \frac{\eta}{\gamma} \left(R_0 + \int_0^t \dot{R}(r)dr \right) - \frac{\eta}{\gamma} R_0 + D_0 && (4.6)\\ &= \frac{\eta}{\gamma} \left(R(t) - R_0 \right) + D_0 \,, && (4.7) \end{aligned}$$

and, since $D_0 = R_0 = 0$ is the initial value (and also in the data set), it follows that

$$D(t) = \frac{\eta}{\gamma} R(t) \,. \tag{4.8}$$

This means that our model, mechanistically, is actually not much different from a regular SIR model (see Section 2.1.1), as it is equivalent to a model that only predicts the three classes S, I, and R', and just subdivides the class R' into the two different classes R and D with a constant and fixed proportion. It follows that it is impossible for this model to fit data in which the data for R and D is not proportional properly. The ratio between R and D in the COVID-19 statistics is drawn in Figure 4.3, where it is immediately clear that not only can the chosen constants not fit the data, but that there are no constants at all that could even approximately work, as R and D are not proportional to each other. Note that this does not mean that the inferred contact rate has to be wrong or bad, but we can not really interpret its quality, as the the model can not explain the data regardless.

Concluding this experiment, we see that our model itself is a bad fit for the data. In order to more accurately explain the data and better infer the contact rate, we need to increase the capacity of the model, by allowing non-constant death and recovery rates. Thus, in the next experiment, we will model the recovery and death rates similarly to the contact rate, and infer all three.

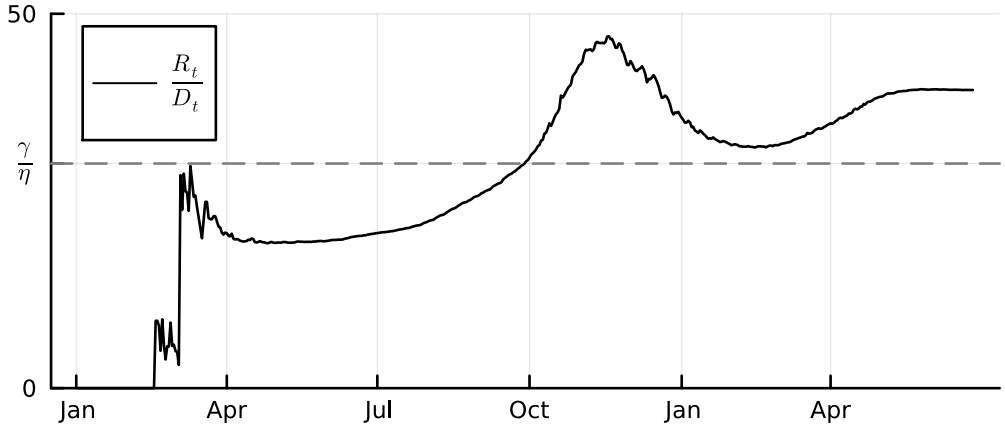


Figure 4.3: **Ratio between recovered and deceased individuals**: Depicted is the ratio between recovered (R_t) and deceased (D_t) individuals for the German COVID-19 statistics. Note that, in the beginning, where $D_t = 0$, the ratio is also set to zero. The dashed grey line is the ratio between the assumed constant recovery rate $\gamma = 0.06$ and death rate $\eta = 0.002$. Clearly, as the ratio between R_t and D_t does not overlap with $\frac{\gamma}{\eta}$, and is itself not constant, a constant death and recovery rate can not fit the data appropriately.

4.3 Inferring the contact, recovery and death rate

As we concluded from the previous experiment, we need to extend our model in order to fit the data properly. In particular, we need to also infer a time-variable recovery and death rate. This experiment evaluates whether we can infer reasonable estimates for the contact, recovery and death rate together, and how well the result can fit the COVID-19 statistics.

4.3.1 Extending the model

In order to infer the recovery and death rate as well, we need to model them as parametric functions. For this, we will just use the same function that we have already been using for the contact rate β as detailed in Section 3.2.2, which is a weighted sum of Gaussian curves, on top of which we apply the sigmoid function.

In advance, we expect the recovery and death rate to not change rapidly and thus be a lot smoother than the contact rate, so we only model them with half the number of parameters as the contact rate ($N_\gamma = N_\eta = \frac{1}{2}N_\beta = 18$), resulting in a total of 72 parameters for all three together. We then also have to widen their individual Gaussian curves, and also set them twice as wide ($\sigma_\gamma = \sigma_\eta = 2\sigma_\beta = 28$), so one individual Gaussian curve has the standard deviation (width) of roughly a month.

The resulting new SIRD equations with a variable contact, recovery and death rate then are given by

$$\begin{aligned} \dot{S}(t) &= -\beta_{\boldsymbol{\theta}} S(t) I(t)/P & \dot{R}(t) &= \gamma_{\boldsymbol{\theta}}(t) I(t) \\ \dot{I}(t) &= \beta_{\boldsymbol{\theta}} S(t) I(t)/P - \gamma_{\boldsymbol{\theta}}(t) I(t) - \eta_{\boldsymbol{\theta}}(t) I(t) & \dot{D}(t) &= \eta_{\boldsymbol{\theta}}(t) I(t)\,. \end{aligned} \tag{4.9}$$

On top of that, we have to choose different prior distributions of the weights. As for the

contact rate, we choose a standard deviation of $\sigma_{w_\gamma} = \sigma_{w_\eta} = 0.01$, but the mean for the recovery rate weights is $\mu_{w_\gamma} = -1.52$, and the mean for the death rate is $\mu_{w_\eta} = -3.43$. This results in prior samples of recovery and death rates similar to the contact rate depicted in Figure 3.1a, but the recovery rate averages at around 0.06, and the death rate around 0.002, the constants that they were previously assumed as. Their initial values are also, like that of the contact rate, sampled from their prior distributions during initialization.

4.3.2 Method

We perform the BFGS algorithm for 1000 iterations with the HagerZhang line search algorithm (as used in the first experiment, Section 4.1) to obtain MLE estimates for the parameters of the contact, recovery and death rates.

4.3.3 Results

The 1000 iterations of the BFGS algorithm took about 95 minutes, with a final error of 84.47409 (rounded). The resulting inferred contact, recovery and death rates, as well as the model predictions for the S, I, R and D counts are depicted in Figure 4.4. The final loss is less than half of what it was in the previous experiment which assumed a constant recovery and death rate, and the overall fit to the observed COVID-19 statistics has improved drastically as well compared to the previous experiment.

4.3.4 Discussion

Evidently, the inclusion of a variable recovery and death rate into the model and inference thereof are necessary for properly fitting the COVID-19 statistics: The resulting model with inferred contact, recovery and death rate can fit the observed data pretty well, which is not the case when inferring the contact rate alone (see previous experiment in Section 4.2).

One still has to be careful with interpreting the resulting inferred rates however. For example, the model inferred a contact rate very close to zero between May and June 2020 (see Figure 4.4), which seems implausible, and has two big spikes before that. These problems might be partly caused by the very low counts of I, R and D before and during that time, and the model overfitting them with large positive or negative weights. Overall though, the resulting contact rate seems to roughly match up with the results obtained by [SKH21], especially so in the second half (after around September 2020). The inferred recovery rate exhibits similar problems in the first months as the contact rate, with a big spike surrounded by two phases with an implausibly low recovery rate. The inferred death rate however looks more plausible, being smooth overall, with two bigger spikes occurring right after big waves of new infections.

Do note however, that this thesis is not meant to provide any sort of realistic conclusions about the data set (the German COVID-19 statistics) itself (or even other things such as governmental measures taken), and should not be considered epidemiologic research. The goal of this thesis is only to learn about the method and model used, therefore, by for example saying that the contact rate fits the data well, we are only stating things within the context of this method and model, and explicitly not claiming relationships to anything outside of that.

Of course, it would be very interesting to have quantifications of the uncertainty about the inferred rates, as well as the prediction of S, I, R and D counts. As explained in Section 2.5.1,

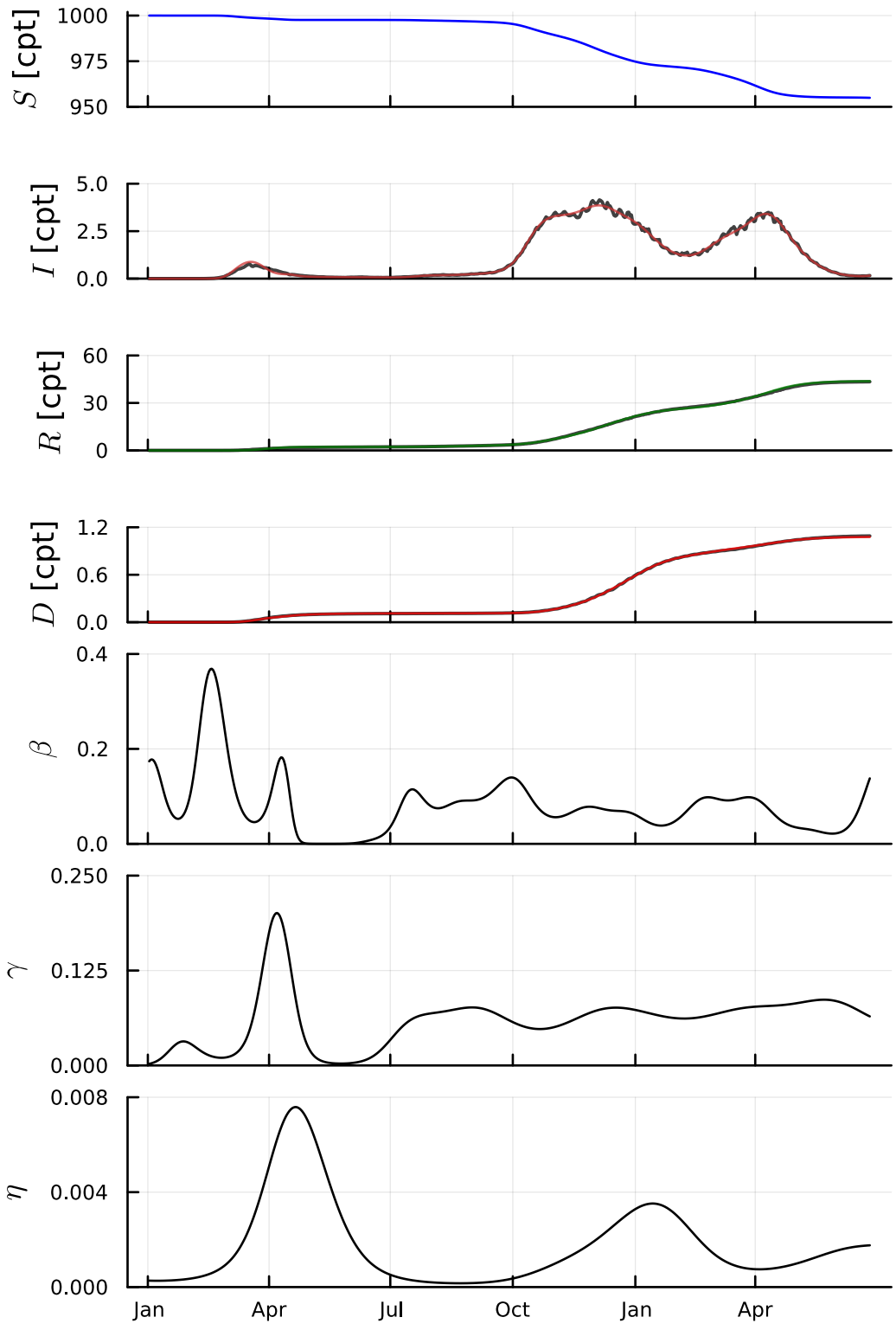


Figure 4.4: **Results of the third experiment**: Shown in the bottom three figures are the MLE estimates for the contact rate β, the recovery rate γ and the death rate η. The top four plots show in color the model predictions for S, I, R and D (rescaled to cases per thousand (cpt)), as well as the observed German COVID-19 statistics in black, for which the contact, recovery and death rates were inferred.

Laplace approximations provide a relatively easy way to obtain uncertainty quantification, which we will use in the next experiment.

4.4 Adding uncertainty quantification to the model

After obtaining a relatively good MLE point estimate for the contact, recovery and death rate of the SIRD model, we will use Laplace approximations in order to obtain a posterior parameter distribution as a measure of uncertainty of the estimated parameters.

4.4.1 The model

The model itself is identical to that used in the last experiment (see Section 4.3), meaning we try to infer parameters of the contact rate and the recovery rate, as well as the death rate. However, the prior parameter distributions that were used to initialize the parameters and had no further use after that, as we just performed MLE, but in this experiment, we also use them as a proper prior distribution of the parameters.

4.4.2 Method

First, we calculate a MAP estimate, as explained in Section 2.3, for the parameters. Since the prior distribution of the parameters is a normal distribution, this boils down to calculating the MLE as before (also explained in Section 2.3), but with an L2 norm parameter penalty, penalizing parameters far from the prior means μ_{w_β}, μ_{w_γ} and μ_{w_η}. The prior parameter distribution standard deviation is set to $\sigma_{w_\beta} = \sigma_{w_\gamma} = \sigma_{w_\eta} = 0.01$ as in the previous experiments, and the model distribution, which is assumed to be lognormal, also has a scale parameter of $\sigma_\epsilon = 0.01$ (as was used in the first experiment, Section 4.1 for data noise). Effectively, this means that the weighting of the MLE loss term and the L2 norm penalty in the loss function for the MAP is equal, but the implementation would also allow for different noise distribution scale parameters and prior standard deviations for a different scaling of the loss and the parameter penalty.

In this experiment, we optimize the loss function in a *two-stage* process. First, for a quick and rough optimization, we run the Adam optimizer for 6000 iterations, with a learning rate of $\lambda = 0.005$. Additionally, for this optimization stage, the ODE solver tolerances are relaxed to 10^{-4}. Then, in the second stage, we tighten the tolerances again to 10^{-5} as in the other experiments, and further optimize with 50 iterations of the Newton algorithm. The line search used with the Newton optimizer is again the HagerZhang line search, with an initial $\alpha = 1$ length. A more extensive reasoning behind this design is given in Appendix A.2

With the MAP estimate obtained through optimization, we can perform the Laplace approximation (introduced in Section 2.5.1), yielding a normal distribution that approximates the posterior parameter distribution. Note that the Hessian $\boldsymbol{H}$ of the loss is likely to not be positive definite, which can happen if the MAP is slightly off from the minimum or due to numerical imprecisions. In this case, we have to calculate a positive definite approximation. This can be done by calculating the Eigendecomposition

$$\boldsymbol{H} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}\,, \tag{4.10}$$

and we can then calculate the approximation

$$\boldsymbol{H}^* = \boldsymbol{V}\boldsymbol{\Lambda}^+\boldsymbol{V}^{-1}\,, \tag{4.11}$$

where the matrix $\mathbf{\Lambda}^+$ with the eigenvalues has all negative (or zero) entries replaced by some small $\epsilon > 0$. In our case, we chose $\epsilon = 10^{-7}$.

After approximating the posterior, we can then sample parameters from this approximation in order to obtain posterior samples for the contact, recovery and death rates. Finally, with these posterior samples, we can obtain a posterior distribution over the counts of S, I, R and D by solving the SIRD equations for every sample.

4.4.3 Results

The two-stage optimization took around four minutes for the first stage with Adam, and around ten minutes for the second stage using the Newton algorithm, for a total of around 14 minutes. The final error was 11503.234 (rounded), of which 1532.598 are due to the regularization term, and 9970.637 come from the MLE loss term. This last MLE loss term can be divided by 100 (the inverse of the model distribution scale factor), resulting in 99.70637, which is comparable to the error from the previous experiment. This error is quite a bit higher than before, which is expected because of the regularization, resulting in a worse overall fit to the data, but also preventing overfitting due to large positive or negative weights.

In Figure 4.5, both the MAP estimate and samples from the Laplace approximated posterior distribution are shown for the contact, recovery and death rates, as well as the predicted S, I, R and D counts for these samples.

4.4.4 Discussion

It seems like the Laplace approximation could work well for uncertainty quantification of our model. In fact, after seeing the posterior samples of the contact, recovery and death rates, it was surprising to us that the posterior samples of the S, I, R and D counts obtained by solving the SIRD equations with the posterior rate samples actually are close to the observed data at all. For the recovered counts R and deceased counts D, the observed data is contained within the area of posterior samples almost everywhere (except for D between February and March 2021). For the infected counts I, the observed data is not within the area of posterior samples everywhere, especially during the first wave of infections in April 2020, and partly the second wave of infections around December 2021. However, the distance between posterior samples (the variance between the samples) captures the noise in the data rather well, and is small (lower variance) when the data seems to have lower noise, and bigger when the data noise is higher, just fitting most of the observed data points within the area of posterior samples.

However, overall, the Laplace approximation shows a problematically low sample variance, or, in other words, the approximation is overconfident, even at time points where we would expect higher uncertainties, such as for example in the first three months, where infection counts are very low. This is also the time span of lowest uncertainty in the contact, recovery and the death rate. Also, it is important to note that we did not perform any model selection or hyperparameter optimization for the Laplace approximation. Most importantly, at least the model distribution width parameter σ_ϵ and the prior distribution would have to be optimized as well in order to obtain any kind of interpretable and calibrated posterior distribution (and with that, a quantification of uncertainty), which we did not do. Thus, the posterior distribution we obtained in this experiment is not really meant to be interpreted as a good posterior distribution, and far from a definitively good approximation. We only tried the given values and, again, did not perform any hyperparameter optimization. The intent is just to

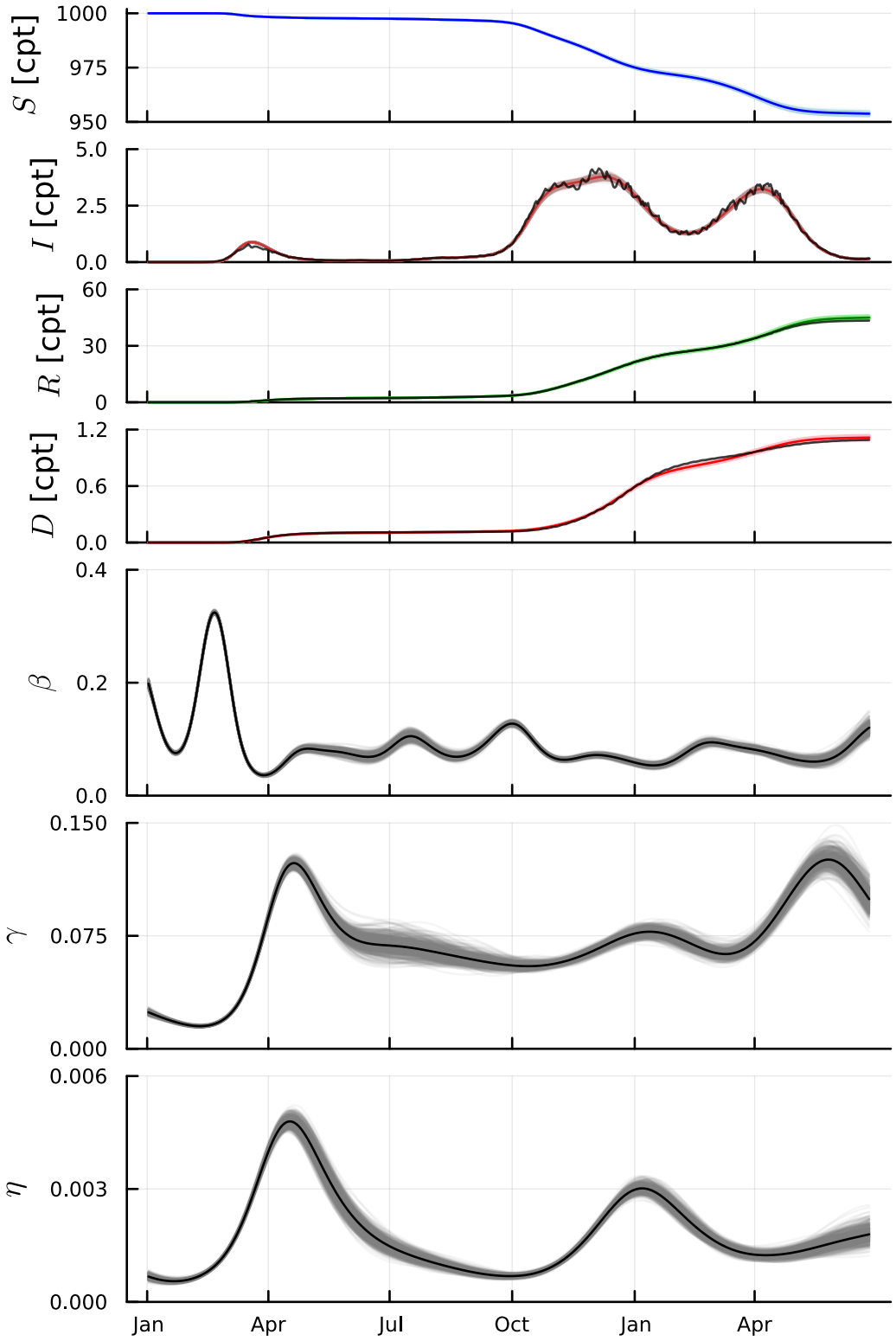


Figure 4.5: **Results of the fourth experiment**: Depicted are the MAP estimates (black lines for the contact rate β, recovery rate γ and death rate η, darker colored lines for the S, I, R and D counts), as well as 500 samples from the approximated posterior distribution (grey lines for the contact, recovery and death rates, light colored lines for the S, I, R and D counts). The observed German COVID-19 counts for I, R and D are depicted in black, and the counts for S, I, R and D have been rescaled to cases per thousand (cpt). Regions where samples are spread further apart (higher sample variance) indicate a higher uncertainty.

show that, for this model, a Laplace approximation of the posterior has interesting properties that reflect the observed data, and it *could* be used as a proper measure of uncertainty, if the hyperparameters are carefully selected and thus the approximation calibrated. Furthermore, Laplace approximations are effective and easy to perform in this setting, with a very low additional computational cost.

Also worthy of note is the regularizing effect of the prior distribution for the MAP estimate. The inferred contact, recovery and death rates are a lot smoother as compared to MLE estimate in the previous experiment (see Section 4.3). On top of that, other implausible aspects like the near-zero contact rate between May and June 2020, which were caused by large negative weights, are not present or softened in the MAP estimate due to the regularization.

Finally, the two-stage training process has shown to be very effective, and did not only drastically reduce the training time (as compared to the third experiment, in which we just used BFGS) by almost an order of magnitude, but as required for the Laplace approximation, yielded a MAP very close to a local minimum, with an approximately zero gradient and approximately positive definite Hessian.

Chapter 5

Conclusion

In this thesis, we have shown that a SIRD model inferring the contact rate, but with fixed recovery and death rate, can not accurately fit the German COVID-19 statistics. Instead, the recovery and death rate have to be inferred as latent forces as well. When extending the model by inferring the contact, recovery and the death rate, we can obtain reasonable estimates for the contact, recovery and death rates and fit the data reasonably well.

Overall, our inferred contact rate shows similarities to what [SKH21] obtained with their approach using probabilistic state space models, although our quantification of uncertainty using Laplace approximations is very rough. We have also demonstrated an advantage of iterative point estimate optimization, which is the fact that the point estimate is directly interpretable, and easily evaluated. With the obtained MLE point estimate in the second experiment (Section 4.2), it is immmediately clear that the model can not fit the data properly with a constant recovery and death rate, a fact that could remain hidden or would be harder to spot if we only had posterior distributions available.

On top of that, Laplace approximations present an easy and computationally inexpensive way to quantify the model uncertainty. Although we did not achieve calibrated or well-interpretable uncertainties, we were still able to obtain meaningful posterior samples with a good fit to the data. As already mentioned in the last experiment, one possible improvement would be the hyperparameter selection (for example of the prior and the model distribution), which could be improved in future work, and also performed using the Laplace approximation. The purpose of this would be to achieve the calibration of the uncertainties that is still missing, and perhaps also obtaining a well-interpretable posterior distribution.

The complete inference process can, with careful selection of hyperparameters, be completed within minutes, or sometimes, if it works, within seconds (see the second experiment in Section 4.2). However, the whole process and its computational demands are heavily dependent on a multitude of factors, such as the number of parameters, the chosen optimizer, and more, with time demands spanning ranges from seconds to potentially hours. Furthermore, by tweaking solver tolerances, there is a further potential for speedup at the cost of accuracy (see Appendix A.1) that can be considered.

Another open question would be for example how the approach taken in this thesis would translate over to other models for the contact, recovery and death rates, such as for example to a standard neural network architecture predicting all three rates.

Lastly, it would be interesting to see how well MCMC methods would perform in this context. According to [SKH21], just inferring the contact rate alone would take time in the range of multiple hours, and it would be interesting to see how well they would work for inferring the contact, recovery and the death rate, and how computationally expensive it would be. Overall, the iterative optimization approach we used in this thesis performed a lot worse than the approach introduced by [SKH21] using probabilistic state space models. Even though we discussed an advantage of the obtained point estimate, iterative optimization was still way more computationally expensive, and all potential speed improvements we presented come with significant loss of accuracy.

Bibliography

[BAB+19] Nathan Baker, Frank Alexander, Timo Bremer, Aric Hagberg, Yannis Kevrekidis, Habib Najm, Manish Parashar, Abani Patra, James Sethian, Stefan Wild, Karen Willcox, and Steven Lee. Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence. 2 2019.

[BEKS17] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[BGJM11] Steve Brooks, Andrew Gelman, Galin L. Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. Handbooks of Modern Statistical Methods. Chapman & Hall/CRC, 2011.

[Bis94] Christopher M. Bishop. Mixture density networks. Workingpaper, Aston University, 1994.

[Bro70] C. G. Broyden. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 03 1970.

[But08] John C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2nd ed edition, 2008.

[CG05] Fabio Clementi and Mauro Gallegati. Pareto's Law of Income Distribution: Evidence for Grermany, the United Kingdom, and the United States. Microeconomics 0505006, University Library of Munich, Germany, May 2005.

[CRBD19] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.

[CSR+23] Simon Christ, Daniel Schwabeneder, Christopher Rackauckas, Michael Krabbe Borregaard, and Thomas Breloff. Plots.jl – a user extendable plotting api for the julia programming language. 2023.

[DDG20] E. Dong, H. Du, and L. Gardner. An interactive web-based dashboard to track COVID-19 in real time. *Lancet Infect Dis*, 20(5):533–534, May 2020.

[DKI+22] Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux – effortless bayesian deep learning, 2022.

[Eul68] L. Euler. *Institutionum calculi integralis*. Number Bd. 1 in Institutionum calculi integralis. imp. Acad. imp. Saènt., 1768.

[Fle70] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 01 1970.

[GBB+20] Giulia Giordano, Franco Blanchini, Raffaele Bruno, Patrizio Colaneri, Alessandro Di Filippo, Angela Di Matteo, and Marta Colaneri. Modelling the covid-19 epidemic and implementation of population-wide interventions in italy. *Nature Medicine*, 26(6):855–860, Jun 2020.

[Gol70] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[Het00] Herbert W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42(4):599–653, 2000.

[HOK22] Philipp Hennig, Michael A. Osborne, and Hans P. Kersting. *Probabilistic Numerics: Computation as Machine Learning*. Cambridge University Press, 2022.

[HZ06] William W. Hager and Hongchao Zhang. Algorithm 851: Cg_descent, a conjugate gradient method with guaranteed descent. *ACM Trans. Math. Softw.*, 32(1):113–137, mar 2006.

[IBF+21] Alexander Immer, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Mohammad Emtiyaz Khan. Scalable marginal likelihood estimation for model selection in deep learning, 2021.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[Kon77] K. Kondo. The lognormal distribution of the incubation time of exogenous diseases. Genetic interpretations and a computer simulation. *Jinrui Idengaku Zasshi*, 21(4):217–237, Mar 1977.

[LPB17] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles, 2017.

[LSA01] Eckhard Limpert, Werner A. Stahel, and Markus Abbt. Log-normal Distributions across the Sciences: Keys and Clues: On the charms of statistics, and how mechanical models resembling gambling machines offer a link to a handy way to characterize log-normal distributions, which can provide deeper insight into variability and probability—normal or log-normal: That is the question. *BioScience*, 51(5):341–352, 05 2001.

[LWB+19] Dahua Lin, John Myles White, Simon Byrne, Douglas Bates, Andreas Noack, John Pearson, Alex Arslan, Kevin Squire, David Anthoff, Theodore Papamarkou, Mathieu Besançon, Jan Drugowitsch, Moritz Schauer, and other contributors. JuliaStats/Distributions.jl: a Julia package for probability distributions and associated functions, July 2019.

[LZG+20] Qianying Lin, Shi Zhao, Daozhou Gao, Yijun Lou, Shu Yang, Salihu S. Musa, Maggie H. Wang, Yongli Cai, Weiming Wang, Lin Yang, and Daihai He. A conceptual model for the coronavirus disease 2019 (covid-19) outbreak in wuhan, china with individual reaction and governmental action. *International Journal of Infectious Diseases*, 93:211–216, 2020.

[Mac92] David J. C. MacKay. Bayesian Interpolation. *Neural Computation*, 4(3):415–447, 05 1992.

[MR18] Patrick Kofod Mogensen and Asbjørn Nilsen Riseth. Optim: A mathematical optimization package for Julia. *Journal of Open Source Software*, 3(24):615, 2018.

[NW99] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 1999.

[Pol64] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, USA, 3 edition, 2007.

[RBB18] Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *International Conference on Learning Representations*, 2018.

[RLP16] J. Revels, M. Lubin, and T. Papamarkou. Forward-mode automatic differentiation in Julia. *arXiv:1607.07892 [cs.MS]*, 2016.

[RMM+21] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. Universal differential equations for scientific machine learning, 2021.

[RN17] Christopher Rackauckas and Qing Nie. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *The Journal of Open Research Software*, 5(1), 2017. Exported from https://app.dimensions.ai on 2019/05/05.

[RPK17] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations. *CoRR*, abs/1711.10561, 2017.

[RPK19] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[Sar50] P. E. Sartwell. The distribution of incubation periods of infectious disease. *Am J Hyg*, 51(3):310–318, May 1950.

[Sar52] P. E. Sartwell. The incubation period of poliomyelitis. *Am J Public Health Nations Health*, 42(11):1403–1408, Nov 1952.

[Sar66] P. E. Sartwell. The incubation period and the dynamics of infectious disease. *Am J Epidemiol*, 83(2):204–206, Mar 1966.

[Sha70] David F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24:647–656, 1970.

[SKH21] Jonathan Schmidt, Nicholas Krämer, and Philipp Hennig. A probabilistic state space model for joint inference from differential equations and data. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12374–12385. Curran Associates, Inc., 2021.

[TH+12] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[TKSH19] Filip Tronarp, Hans Kersting, Simo Särkkä, and Philipp Hennig. Probabilistic solutions to ordinary differential equations as non-linear bayesian filtering: A new perspective, 2019.

Appendix A

On the choice of hyperparameters

A.1 ODE solver tolerances

We found the tolerances for the ODE solver, in our case, a Rosenbrock method based solver, to heavily influence both the training time, as well as the results. Obviously, for higher tolerances, the training process is much, much faster, but it also results in significantly worse results (higher final error). This enables one to explicitly trade off computational cost for accuracy. For the setup in the first experiment (see Section 4.1), we have tried out different tolerances in order to emphasize this, the results are shown in Table A.1. Based on these results, we opt for a solver tolerance (both absolute and relative) of 10^{-5}, which seems to provide the best trade-off between execution time and accuracy.

A.2 Optimization algorithms

This section describes problems with the choice of optimization algorithms. It applies to all experiments, but is written in the context of the fourth, as it is the only one where we really need a good enough minimum.

One big problem of this thesis is the selection of optimal optimization algorithms. Particularly, it is very hard for the optimizer to *converge* in our setting, by which we mean terminating at least very close to a minimum, and therefore an approximately zero gradient, and a positive definite Hessian. While for the first three experiments, it is enough to get reasonable estimates

Tolerance	10^{-3}	10^{-4}	10^{-5}	10^{-6}
Final error	6.6	3.0	1.64	1.56
Optimization time	12 seconds	30 seconds	9 minutes	$>$ 49 minutes (aborted)
Iterations	77	77	397	$>$ 700 (did not converge)

Table A.1: **Impact of different solver tolerances on optimization result and duration**: This table contains the final error (rounded to two decimal places), as well as how much time and how many iterations of optimization were required, with the setup of the first experiment (Section 4.1), but choosing different ODE solver tolerances. A tolerance of 10^{-5} was chosen for this thesis.

with low error, the Laplace approximation performed in the fourth experiment (Section 4.4) requires that the estimate is a minimum (or, technically, a maximum of the joint prior and likelihood distribution), and even small deviations from it become problematic.

For example, Adam can reduce the error quickly and get close to minima, but no matter how small a (realistic) learning rate we choose, it does not converge. Even with a learning rate of for example 10^{-5}, Adam starts to oscillate after about 180000 iterations.

With BFGS, we have the two main options for line search, HagerZhang line search and backtracking line search. Here, the former produces fairly consistent results, but requires far more function evaluations, which are very expensive since they involve solving an ODE every time. With backtracking line search, we can often very quickly (orders of magnitude faster, usually in the range of seconds) obtain results, but they very often with implausibly large weights and generally suboptimal (in the sense that other optimization algorithms almost always obtained results with significantly lower loss).

For the fourth experiment, we opt to use the Newton optimization algorithm, which is even more expensive, but works with the (still numerically approximated) true Hessian instead of the approximation that BFGS uses. Therefore, it should be more accurate, and we found it to be more consistent. In order to save potentially lots of time, we use Adam in a first stage, until about where it starts to oscillate (which, in the fourth experiment, are about 6000 iterations), and then further optimize the result of that in a second stage, using the Newton algorithm. Note that, since Adam does not have to optimize the minimum accurately, we can relax the solver tolerances in the first stage (as stated in the previous section).

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Felix Böhm (Matrikelnummer 5443299), 02. Mai 2023