Accepted author's manuscript. Published in final edited form as: Statistics in Medicine 2021 (in press).
Publisher DOI: 10.1002/sim.9164

Bayesian workflow for disease transmission modeling in Stan

Léo Grinsztajn[*], Elizaveta Semenova[†], Charles C. Margossian[‡], and Julien Riou[§]

Abstract

This tutorial shows how to build, fit, and criticize disease transmission models in Stan, and should be useful to researchers interested in modeling the SARS-CoV-2 pandemic and other infectious diseases in a Bayesian framework. Bayesian modeling provides a principled way to quantify uncertainty and incorporate both data and prior knowledge into the model estimates. Stan is an expressive probabilistic programming language that abstracts the inference and allows users to focus on the modeling. As a result, Stan code is readable and easily extensible, which makes the modeler's work more transparent. Furthermore, Stan's main inference engine, Hamiltonian Monte Carlo sampling, is amiable to diagnostics, which means the user can verify whether the obtained inference is reliable. In this tutorial, we demonstrate how to formulate, fit, and diagnose a compartmental transmission model in Stan, first with a simple Susceptible-Infected-Recovered (SIR) model, then with a more elaborate transmission model used during the SARS-CoV-2 pandemic. We also cover advanced topics which can further help practitioners fit sophisticated models; notably, how to use simulations to probe the model and priors, and computational techniques to scale-up models based on ordinary differential equations.

Keywords: infectious diseases, compartmental models, epidemiology, Bayesian workflow

1 Introduction

The pandemic of severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) has led to a renewed interest in infectious disease modeling and, amongst other approaches, Bayesian modeling. Well-constructed models allow researchers to infer the value of key epidemiological parameters required to inform public health policies. Over the early part of the pandemic, modelers have assessed the effect of control interventions on transmission [1], quantified the burden of the epidemic [2], and estimated the mortality rate after adjusting for reporting biases[3], to only name a few examples.

Mechanistic disease transmission models mirror natural phenomena such as contagion, incubation and immunity [4]. These models can operate at different scales, the most important being the individual scale (individual- or agent-based models) and the population scale (population-based or compartmental models). Agent-based models are used to simulate the occurrence of stochastic events, such as transmission or symptom onset, in time for every individual of the population. These models can reach a high level of sophistication but are computationally expensive. Agent-based models are also notoriously difficult to parametrize.

Population-based or compartmental models subdivide the total population into homogeneous groups, called compartments. Individuals within a compartment are considered to be in the same state with regards to the natural history of the disease. These states may for example be "susceptible", "infectious", and "recovered". We can model transitions between compartments as deterministic or stochastic. In the deterministic framework, the flows between compartments can be described by a system of ordinary differential equations (ODEs). This implies that the size of each compartment at every time point can be obtained by numerically solving the system of ODEs, with the same parametrization always leading to the same outcome. The flows between compartments can also be simulated stochastically, leading to the same results on average over multiple simulations (ignoring disease extinctions) but with a better handling of uncertainty that comes at a generally higher computational price. This makes stochastic compartmental models more adapted to low-level transmission and small populations. In other situations, deterministic compartmental models are easier to formulate and computationally tractable, which makes them more adapted to tasks that require simulating the system a large number of times, such as model fitting. This article focuses on deterministic compartmental models.

`Stan` is a probabilistic programming framework primarily used for Bayesian inference and designed to let the user focus on modeling, while inference occurs under the hood [5]. One goal of the `Stan` project is not merely to improve model fitting but to make model development more efficient. This development process includes building, debugging, improving, and expanding models as more data and knowledge become available, and motivates the concept of the *Bayesian modeling workflow* [6, 7, 8], a central topic in this article. `Stan` is an expressive language that supports many probability densities, matrix operations, and numerical ODE solvers. We can combine these elements to specify a broad range of data generating processes. Generative models formulated in `Stan` can be used both for simulation-based prediction and for parameter inference. In the context of epidemiological modeling, the powerful framework can help us estimate such crucial parameters as the basic reproduction number $\mathcal{R}_0$ or the infection-fatality ratio from observed data. `Stan` bolsters several inference methods: full Bayesian inference using Markov Chain Monte Carlo (MCMC), approximate Bayesian inference with variational inference, and penalized maximum likelihood estimation. We focus here on Bayesian

[*]École polytechnique, Palaiseau, France, `leo.grinsztajn@polytechnique.edu`
[†]Data Sciences and Quantitative Biology, Discovery Sciences, R&D, AstraZeneca, Cambridge, UK
[‡]Department of Statistics, Columbia University, New York, NY, USA
[§]Institute of Social and Preventive Medicine, University of Bern, Bern, Switzerland

source: https://doi.org/10.48350/159351 | downloaded: 26.4.2024

inference with MCMC, specifically with the dynamic Hamiltonian Monte Carlo (HMC) sampler [9, 10]. Bayesian inference gives us a principled quantification of uncertainty and the ability to incorporate domain knowledge in the form of priors, while HMC is a reliable and flexible algorithm. In addition, `Stan` provides diagnostic tools to evaluate both the reliability of the inference and the adequacy of the model.

This tutorial examines how to formulate, fit, and diagnose compartmental models for disease transmission in `Stan`. We first focus on a simple Susceptible-Infected-Recovered (SIR) model, before tackling a more sophisticated model of SARS-CoV-2 transmission. Rather than only show the results from a polished model, we devote much attention to dealing with flawed models and flawed inference; and we explore how different modes of failure can help us develop improved models and better tune our inference algorithms. Naturally, `Stan` is not the only tool available to practitioners and many of the concepts we discuss can be deployed using other probabilistic programming languages. At times, we dig into the specific mechanics of `Stan` in order to provide a more practical discussion on modeling and computational efficiency. A complementary notebook[11] with the full code, and a Github repository*containing the relevant scripts are available online. Throughout the tutorial, we use `R` as a scripting language (`Stan` can also be used with other languages such as Python, Julia or Matlab). While we review some elementary concepts, we assume that the reader has basic familiarity with Bayesian inference and Stan. Other tutorials on the subject include the work by Chatzilena et al.[12] and Mihaljevic [13] on transmission models, and the case studies by Carpenter [14], Weber [15], and Margossian and Gillespie [16] on ODE-based models, all of which can serve as complementary reading.

2 Bayesian modeling in Stan

`Stan` is a tool which provides both a language to formulate probabilistic models and methods to do inference on these models. Before using `Stan` to model disease transmission, we quickly review how to specify a probabilistic model. A more thorough introduction to the topic can be found here [17].

2.1 Specifying a model

We can specify a Bayesian model by defining a joint distribution over observed variables, $\mathcal{Y}$, and unobserved variables, θ,

$$p(\mathcal{Y}, \theta).$$

In this tutorial, θ is simply the set of unknown model parameters. A `Stan` file defines a procedure to evaluate the log density, $\log p(\mathcal{Y}, \theta)$. This joint distribution conveniently decomposes into two terms - the *prior density* $p(\theta)$ and the *sampling density* (or *likelihood*) $p(\mathcal{Y} \mid \theta)$, defining a generating process for $\mathcal{Y}$ given parameters θ :

$$p(\mathcal{Y}, \theta) = p(\theta)p(\mathcal{Y} \mid \theta).$$

2.2 Bayesian inference

Inference reverse-engineers the data generating process and aims to estimate parameter values given the observations. In a Bayesian framework, the set of plausible parameter values conditional on the data is characterized by the *posterior distribution*. The posterior distribution combines information from the data and prior knowledge and is obtained via Bayes' rule

$$p(\theta \mid \mathcal{Y}) \propto p(\theta)p(\mathcal{Y} \mid \theta).$$

An analytical expression for $p(\theta \mid \mathcal{Y})$ is rarely available and we must rely on inference algorithms to learn about the posterior distribution. One general strategy is to draw approximate samples from the posterior distribution and use these to construct sample estimates of the posterior mean, variance, median, quantiles, and other quantities of interest. This leads to a general class of algorithms called Markov chain Monte Carlo (MCMC) samplers.

Hamiltonian Monte Carlo. `Stan` supports dynamic Hamiltonian Monte Carlo [10, 18] (HMC), a widely popular MCMC method. HMC exploits the gradient of $\log p(\theta, \mathcal{Y})$ to simulate trajectories, with acceleration informed by the local geometry of the posterior density. This leads to a rapid exploration of the parameter space and reduces correlation between successive samples. HMC has been shown to scale better than random walk samplers, e.g. Metropolis and Gibbs, when exploring high-dimensional spaces or when parameters exhibit a strong posterior correlation[9]. The method has been successfully applied across a broad range of problems. When HMC fails, it typically does so "loudly", meaning diagnostics can reliably determine whether or not the inference should be trusted. While the method was first proposed in 1987[19], its challenging implementation prevented its wide adoption by the statistics and scientific community. Indeed, calculating the gradient of $\log p(\theta, \mathcal{Y})$ is a cumbersome task and the original HMC algorithm entails many tuning parameters which, without proper tuning, result in suboptimal performance. The advent of automatic differentiation[20, 21] and of the No-U-Turn sampler, an adaptive HMC algorithm[9], in large parts resolved these problems, making it straightforward to apply the algorithm across a broad range of models. With a software such as `Stan`, it is indeed possible to revise a model without needing to rewrite the HMC sampler. In summary, HMC works well in high-dimensional spaces, can handle the intricate posterior geometry that arises in sophisticated models, can be readily applied to several models, and is amiable to diagnostics. For these reasons, we find HMC to be very well suited for a Bayesian workflow (Section 3). Naturally, the algorithm is not without limitation:

*`https://github.com/charlesm93/disease_transmission_workflow`

for example, HMC cannot be applied to discrete problems without marginalization or continuous relaxation. Furthermore, densities with difficult geometries (e.g. multiple modes, heavy tails, varying scales) can frustrate the algorithm[22][23] and, while being broadly applicable, HMC can be slower than specialized algorithms used for certain statistical models.

2.3 Getting started with Stan

Installation At its core, `Stan` is written in `C++` but for convenience it can be interfaced with a scripting language, such as `R` and `Python`; see the Stan interface page[24] for instructions on how to install these interfaces. This tutorial uses `RStan` and `R`.

Learning Stan While we do not assume the reader is already familiar with `Stan`, we encourage them to consult one of the many resources available to learn the language. As a starting point, we recommend the introduction by Betancourt[25]. The reference manual[26] documents the language's syntax; the Stan user guide[27] provides the code for numerous models. Guidelines on how to run `Stan` and various diagnostics are available on the Stan interface page[24]. A complementary notebook[11] offers a detailed walk-through of the example Section 4 and can be used as a hands-on way to learn the language.

Going forward To learn more, the reader may find it helpful to read some of the cases studies[28] and tutorials[29] on specific applications of `Stan`. We also recommend the Stan forum as a place where modelers can discuss their models with other researchers.

2.4 Coding a model in Stan

Using a `Stan` file, we can specify a model in a way that is suitable to Bayesian inference. This entails (i) defining a procedure to compute the log joint density, $\log p(\mathcal{Y}, \theta)$, and (ii) identifying which variables are fixed data, $\mathcal{Y}$, and which ones are latent variables, θ. For pedagogical and computational reasons, a `Stan` file is divided into *coding blocks*, which help us distinguish the different components of our model. Figure 1 provides a description of each block and indicates the order in which the blocks are executed. The three main blocks are `data`, `parameters`, and `model`. In the `data` block, we declare the fixed variables, $\mathcal{Y}$, and in the `parameters` block the model parameters, θ. In the `model` block, the user specifies a set of operations on the declared variables to compute the log joint density, $\log p(\mathcal{Y}, \theta)$. It is possible to do further operations on the fixed data and on the model parameters by using the `transformed data` and the `transformed parameters` blocks, respectively. Additional quantities which depend on the parameters, but are not required to compute the log joint density – for example, predictions – can be computed in the `generated quantities` block. Finally, the `function` block can be used to declare functions which we can call in all the operative blocks, i.e. all blocks but `data` and `parameters`.

Section 5 offers more details on `Stan`'s inference engine and discusses how each block interacts with the inference algorithm. For now, we simply note that the computational cost of fitting a model is dominated by operations in the `transformed parameters` and `model` blocks.

3 Bayesian modeling workflow

The notion of a modeling workflow has likely existed in one form or the other for quite some time. One useful illustration of this concept is Box's loop [30, 31]. The loop prescribes the following workflow: build a model, fit the model, criticize, and repeat. We find it useful to distinguish three parts in the criticism step: (i) troubleshoot the model before fitting it, (ii) criticize the inference after attempting a fit, and (iii) once the inference is deemed reliable, criticize the fitted model (Figure 2). The goal of the criticism is to identify shortcomings in our methods and motivate adjustments, which is why the process loops back to the first and second steps.

The first takeaway from this concept is that model development is an iterative process: we do not expect the first model we devise to be perfect and are prepared to make adjustments. Potential limitations in our model can include implementation errors, inappropriate priors or failure to account for a crucial step in the data generating process. It is essential to detect and correct these problems before we apply the model. Several points can make this potentially tedious process easier. First, our probabilistic programming language must be expressive enough to accommodate our initial model and subsequent revisions. Second, we prefer efficient and flexible inference algorithms that can operate quickly and do not require tuning each time we revise the model. Third, we require the inference to be reliable, meaning we can diagnose many different types of failures, and hopefully adjust our algorithms or even our model to overcome issues we identify. This is especially important when we care about the posterior distribution of our parameters, as we might in epidemiological models where the parameters have a scientific interpretation. We must realise, however, that even when the inference is reliable, the model may fail to solve the scientific problem at hand.

`Stan` and its HMC sampler typically meet these criteria, and provide solutions and features adapted to each of the goals outlined in our modeling workflow. Predictions computed across prior distributions (prior predictive checks) are useful to detect implementation errors and assess the adequacy of the chosen priors for the problem at hand [6]. `Stan`'s HMC sampler provides multiple inference diagnostics, including the detection of bias in sampling by running multiple MCMC chains and monitoring mixing, energy and divergent transitions. Finally,

`data`

Declare known variables, which remain fixed while running MCMC. This includes covariates, observed responses, and other constants.

No operations to perform.

`transformed data`

Declare additional fixed variables and do operations on variables declared in `data` and in this block. This is useful when we want to work with transformations of our original data or create constants for bookkeeping purposes.

Operations are performed once per model fit.

`parameters`

Declare unknown variables, typically model parameters. When running MCMC this also defines the space over which we run the Markov chain.

No operations to perform.

`transformed parameters`

Run operations on the parameters. This is, for instance, where we will solve differential equations which depend on model parameters.

Operations are performed and differentiated once per integration step, i.e. multiple times per iteration.

`model`

Compute $\log p(\mathcal{Y}, \theta)$, using operations on variables declared in the previous blocks.

Operations are performed and differentiated once per integration step, i.e. multiple times per iteration.

`generated quantities`

Run operations on all previously declared variables to compute quantities of interest, such as derived variables and predictions. The variables we compute here depend on the parameters, but crucially none of the operations in this block contribute to computing $\log p(\mathcal{Y}, \theta)$.

Operations are performed once per iteration.

Figure 1: Coding blocks in a `Stan` file. The operations in certain blocks are performed multiple times and in some cases differentiated; as a result, the computational cost of fitting the model is dominated by the transformed parameters and model blocks. Not shown is the `functions` block, which defines functions that can be called in any of the operative blocks.

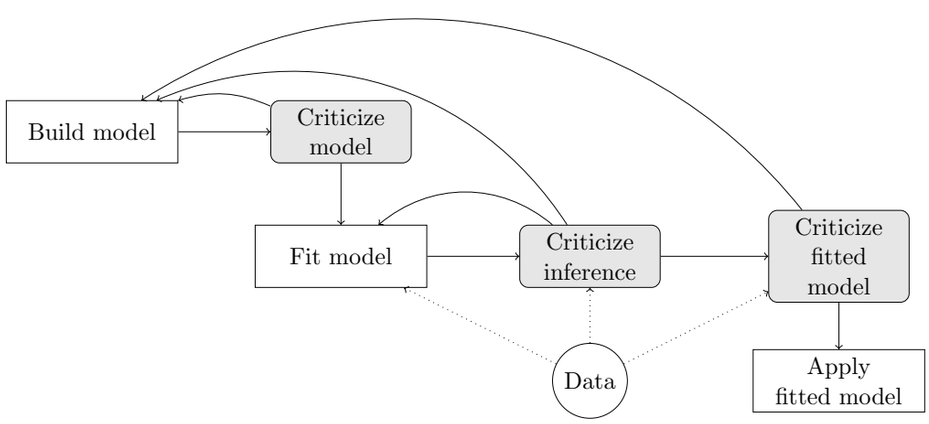

Figure 2: Model development as an iterative process.

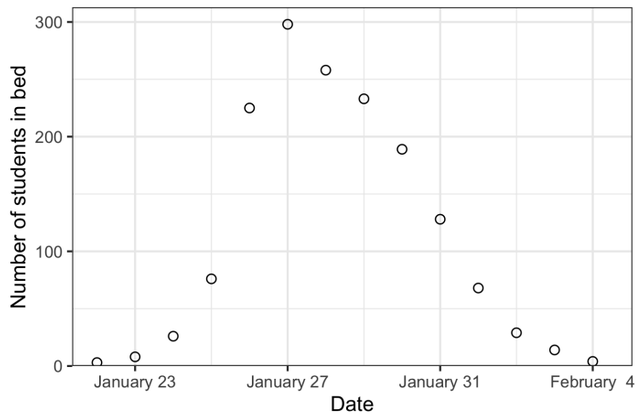

Figure 3: Number of students in bed each day during an influenza A (H1N1) outbreak at a British boarding school between January 22 and February 4, 1978.

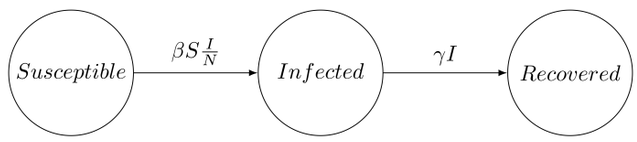

Figure 4: Diagram of the classic Susceptible-Infectious-Recovered (SIR) compartmental model.

we can study the implications of the fitted model by computing predictions across the posterior distribution (posterior predictive checks).

4 Simple SIR model

In this section, we demonstrate how to use `Stan` and the Bayesian workflow on a simple example of disease transmission: an outbreak of influenza A (H1N1) in 1978 at a British boarding school. The data consists of the daily number of students in bed, spanning over a time interval of 14 days, and is displayed in Figure 3. There were 763 male students who were mostly full boarders and 512 of them became ill. The outbreak lasted from the 22nd of January to the 4th of February. One infected boy started the epidemic, which spread rapidly in the relatively closed community of the boarding school. The data are freely available in the R package `outbreaks`[32], maintained as part of the R Epidemics Consortium. The code from this analysis, including the `Stan` model, is available in the notebook online[†], which can serve as a hands-on tutorial to fit your own disease transmission model in Stan.

4.1 Mathematical transmission model

As discussed in section 1, epidemiological transmission models come in various forms, the two main categories being *agent-based models*, which model each individual but are computationally expensive, and *population-based models*, which stratify the population into several homogeneous compartments. This section focuses on one of the simplest population-based models, which is popular for its conceptual simplicity and its computational tractability: the Susceptible-Infected-Recovered (SIR) model. The SIR model splits the population into three time-dependent compartments: the susceptible, the infected (and infectious), and the recovered (and not infectious) compartments. When a susceptible individual comes into contact with an infectious individual, the former can become infected for some time, and then recovers and becomes immune. The dynamic is summarized graphically in Figure 4. The temporal dynamics of the sizes of each of the compartments are governed by the following system of ODEs:

$$\begin{cases} \dfrac{dS}{dt} = -\beta S \dfrac{I}{N} \\ \dfrac{dI}{dt} = \beta S \dfrac{I}{N} - \gamma I \\ \dfrac{dR}{dt} = \gamma I \end{cases}$$

where $S(t)$ is the number of people susceptible to becoming infected (no immunity), $I(t)$ is the number of people currently infected (and infectious), $R(t)$ is the number of recovered people (we assume they remain immune indefinitely), β is the constant rate of infectious contact between people (often called transmission rate) and γ is the rate of recovery of infected individuals (often called recovery rate, but formally relates to the duration of infectiousness rather than to the duration of symptoms).

The proportion of infected people among the population is I/N. At each time step, given uniform contacts, the probability for a susceptible person to become infected is thus $\beta I/N$, with β the average number of contacts per person per time step multiplied by the probability of disease transmission when a susceptible and an infected subject come in contact. Hence, at each time step, $\beta SI/N$ susceptible individuals become infected, meaning

[†] `https://github.com/charlesm93/disease_transmission_workflow`

$\beta SI/N$ people leave the S compartment and enter the I compartment. Similarly, the recovery of an infected individual is taking place at rate γ, and thus the number of infected individuals decreases with speed γI while the number of recovered grows at the same speed.

The above model holds under several assumptions, making it most adapted to large epidemics occurring over relatively short periods of time:

- births and deaths are not contributing to the dynamics and the total population $N = S + I + R$ remains constant,
- recovered individuals do not become susceptible again over time,
- there is no incubation period before becoming infectious,
- the time spent in the infectious compartment follows an exponential distribution with mean $1/\gamma$,
- the transmission rate β and recovery rate γ are constant in time (no behavioural changes nor control measures),
- individuals may meet any other individual uniformly at random (homogeneous mixing),
- replacing the integer number of people in each compartment by a continuous approximation is legitimate since the population is big enough.

4.2 Probabilistic transmission model

We have described a deterministic transmission model, but we want to incorporate randomness into it to model our imperfect knowledge of the model parameters, natural variation and the imperfect fit of the model to reality. As discussed in section 2, we need to specify a sampling distribution $p(\mathcal{Y} \mid \theta)$, and a prior distribution $p(\theta)$, with $\theta = \{\beta, \gamma, 1/\phi\}$ denoting all the parameters of the SIR model plus an additional over-dispersion parameter ϕ that is reparametrized as its inverse (see below). Given specific values for the parameters and initial conditions, a compartmental model defines a unique solution for each of the compartments, including the number of infected students at time t, $I(t)$. We want to link this solution to the observed data, i.e the number of students in bed $\mathcal{Y}$. We choose to model the number of students in bed with a count distribution that provides some flexibility regarding dispersion, the negative binomial distribution. This distribution allows us to use $I(t)$ as the expected value and account for over-dispersion through parameter ϕ:

$$p(\mathcal{Y} \mid \theta) = \text{Negative-binomial}(\mathcal{Y} \mid I(t), \phi).$$

We still need to specify a prior distribution for each of the three parameters using basic domain knowledge. For the transmission rate we select $p(\beta) = \text{Normal}^{+}(2, 1)$ a *weakly-informative prior* that only restricts β to be positive (the half-normal distribution is truncated at 0) and puts a soft higher limit around $4 - p(\beta < 4) = 0.975$. For the recovery rate, we specify $p(\gamma) = \text{Normal}^{+}(0.4, 0.5)$, which expresses our belief that γ has to be positive and that $p(\gamma \leq 1) = 0.9$ (i.e the probability that the average time spent in bed is less than 1 day is 0.9 *a priori*). For the dispersion parameter, we use $p(1/\phi) = \text{exponential}(5)$, as it is recommended in this situation to reparametrize the dispersion parameter as its inverse to avoid putting too much of the prior mass on models with a large amount of over-dispersion [33]. We can change these priors if more information becomes available, constraining our parameter estimation more tightly or, on the contrary, increasing its variance. See [33] for more recommendations on prior choice.

4.3 Building an ODE-based model in `Stan`

Writing this model in `Stan` is very similar to writing it mathematically. As for any `Stan` model, we follow the block structure described in figure 1. We pay particular attention to the `parameters` block, where we declare the model parameters `beta`, `gamma` and `phi_inv` (using an inverse reparametrization for ϕ as discussed above) and their range of support (strictly positive in these cases). We create an array `theta` to hold the parameters together. In the `model` block, we encode the sampling and prior distributions discussed in the previous section. The full code can be found in the complementary notebook. Here, we focus on solving the ODEs, which is the more intricate component of this model. `Stan` supports numerical solvers for ODEs of the form $y' = f(t, y, \vartheta, x)$, where t is a specific time, y a set of ODEs, y' is the solution to the ODE at this time (i.e. the size of each compartment $\{S(t), I(t), R(t)\}$ in our example) and ϑ and x are ODE inputs.

We specify f inside the `functions` block, as follows:

```
real[] f(real t, real[] y, real[] vartheta, real[] x_r, int[] x_i) {
        real S = y[1];
        real I = y[2];
        real R = y[3];
        real beta = vartheta[1];
        real gamma = vartheta[2];
        real N = x_i[1];
        real dS_dt = -beta * I * S / N;
        real dI_dt =  beta * I * S / N - gamma * I;
        real dR_dt =  gamma * I;
        return {dS_dt, dI_dt, dR_dt};
}
```

The function f returns y' as an array of real numbers, as noted by `real[]`. It has five arguments with a specific signature that must be exactly respected. The first two arguments are, as expected, `t` and `y`. The next three

arguments relate to ODE inputs. These are concatenated inside arrays of real or integer numbers. It is crucial to distinguish variables which are model parameters from variables which stay fixed as the sampler explores the parameter space. Here, `vartheta` contains ODE inputs that depend on model parameters θ, that is β and γ, as ϕ is used for the sampling distribution outside of the ODE. Quantities that influence the ODE but remain fixed during the whole procedure (e.g. the population size) are passed using `x_r` and `x_i`, depending on whether they are real or integers,. Distinguishing parameters and fixed variables is a matter of computational efficiency. Section 5 discusses in details the computational cost of solving ODEs in `Stan`. Inside the function, we must declare and define each of the ODEs, but first it is clearer to rename compartments (`S`, `I` and `R`), model parameters (`beta` and `gamma`) and constants (`N`) from the less expressive function arguments.

Having defined the ODE system, we can obtain its approximate solution at any time point of interest by using a numerical solver. In addition to parameters `theta` and constants `x_r` and `x_i`, the solver requires the initial time `t_0`, the initial conditions `y_0` and the different time points `ts` at which a solution is needed. For this example, we use a Runge-Kutta integrator. The call to the numerical solver is:

```
real[,] y = integrate_ode_rk45 (function f, real[] y0, real t0, real[] ts,
                                real[] theta, real[] x_r, int[] x_i);
```

The integrator returns a two-dimensional array with the solution of each compartment at each time point in `ts`. Note that we pass to the integrator the arguments `theta`, `x_r`, and `x_i`, which are in turn passed to the function `f` each time the integrator is being called. Note that as of `Stan` version 2.25, we can relax the signature of `f` and replace `theta`, `x_r`, and `x_i` with any convenient combination of arguments [34].

4.4 Criticizing the model before looking at the data

Before fitting the model to the data, it is useful to check that what we have encoded into the model is indeed coherent with our expectations. This is especially useful for complex models with many transformations. We can check if our priors are sound and correspond to domain knowledge by computing the *a priori* probability of various epidemiological parameters of interest. For instance for influenza, it is generally accepted that the basic reproduction number $\mathcal{R}_0$ is typically between 1 and 2, and that the recovery time is approximately 1 week. We want priors that allow for every reasonable configuration of the data but exclude patently absurd scenarios, per our domain expertise. To check if our priors fulfill this role, we can do a *prior predictive check* [6]. More precisely, for any quantity of interest $\mathcal{Y}$, we can sample $\mathcal{Y}_{\text{prior}}$ from the *a priori* distribution of θ, using the following sequential procedure:

$$\begin{aligned} \theta_{\text{prior}} &\sim p(\theta), \\ \mathcal{Y}_{\text{prior}} &\sim p(\mathcal{Y} \mid \theta_{\text{prior}}). \end{aligned}$$

As demonstrated in the notebook[11], doing so in `Stan` is very simple, as one can use almost the same model implementation that is used for inference. Figure 5A shows the distribution of the log of the recovery time, with the red lines showing loose bounds on the recovery time (0.5 and 30 days). We observe that most of the probability mass is between the red bars but we still allow more extreme values, meaning our posterior can concentrate outside the bars, if the data warrants it. Figure 5B shows the same thing for $\mathcal{R}_0$, the loose bounds being 1 and 10 (note that for the SIR model, $\mathcal{R}_0 = \beta/\gamma$), and figure 5C for the dispersion parameter ϕ.

Another aspect of prior predictive checking is to simulate fake data. Using a similar procedure, we can simulate a set of ODE outputs compatible with the chosen prior distributions of parameters (Figure 5D), or, by adding the variability from the negative binomial distribution, the predicted range of the number of students in bed each day (Figure 5E). It appears that the simulated trajectories are on the same magnitude as the data but still quite diverse, with some epidemics spreading quickly to almost every student and keeping most in bed for two weeks, and on the opposite several delayed or short-lived epidemics. Looking at predicted range of students in bed, we observe that in some extreme situations this number can exceed the total number of students at the boarding school of 763. This is caused by the fact that, if the SIR takes into account the population size, the negative binomial distribution that we put on top of it does not. The prior predictive check often allows to detect this type of non-immediately obvious behaviors from the model. In this case it should not have any consequence.

Put together, the prior predictive checks show that the model is not excessively constrained by the priors, and is still able to fit a wide variety of situations and data. In fact, these priors are likely too wide and encompass scenarios that are impossible or extremely unlikely. Typically, we can get away with priors that do not capture all our *a priori* knowledge, provided the data is informative enough. However when dealing with complicated models and relatively sparse data, we usually need well-constructed priors to regularize our estimates and avoid non-identifiability. We conclude from the prior predictive checks that our choice of prior distributions is adequate.

4.5 Fitting the model

Once the model is written in `Stan` code, the fitting procedure is straightforward from `R` using package `rstan`, or from one of the other available interfaces. Using the default settings, we run 4 independent Markov chains, initialized at different random points, with a warm-up phase that lasts 1,000 iterations and a sampling phase that runs for another 1,000 iterations. There is no definitive rule about the number of chains and iterations. The rule of thumb is to use the default settings as a starting point, and adapt these if needed after criticizing the inference. An important point with regards to computing time is that the chains can be run in parallel, so it

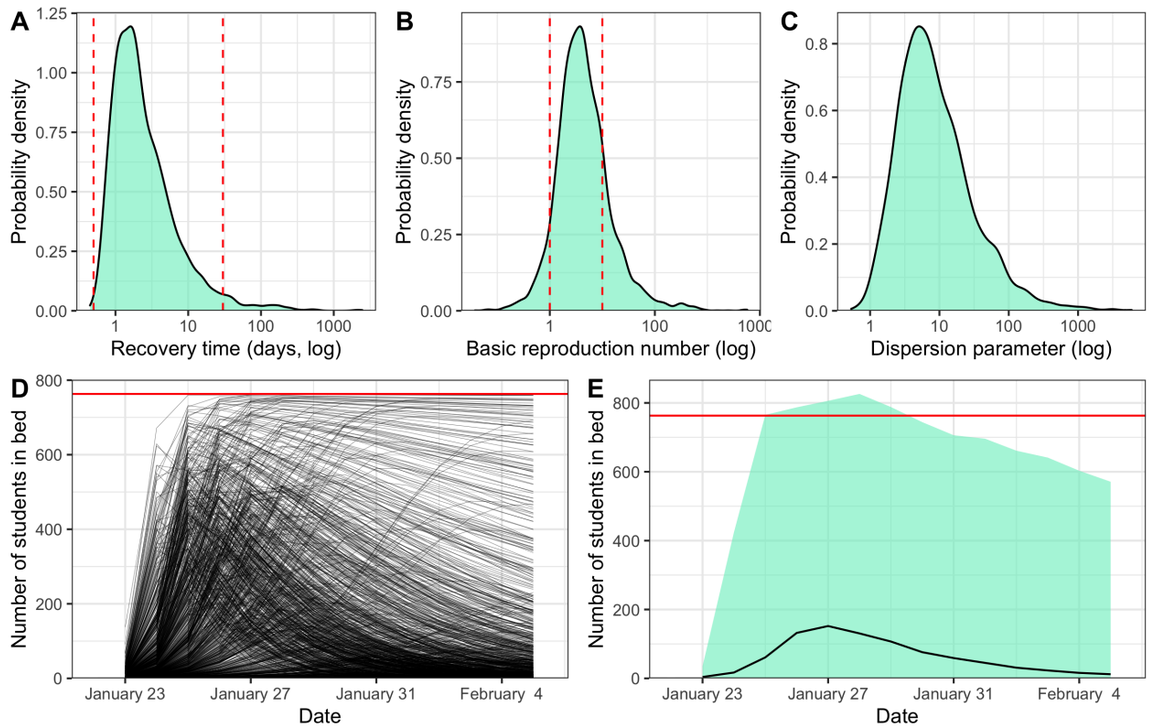

Figure 5: Prior predictive checks for (A) the recovery time ($1/\gamma$), (B) the basic reproduction number $\mathcal{R}_0$ (β/γ), (C) the dispersion parameter (ϕ), (D) a set of 1,000 epidemic trajectories (each line is a unique simulated trajectory) and (E) the range in the numbers of students in bed each day (the line is the median and the light teal area is the 95% central range). All these quantities are sampled from the prior distributions of the parameters. The dashed red lines correspond to weak bounds from our domain knowledge where available: the recovery time is expected to last between 0.5 and 30 days, $\mathcal{R}_0$ cannot be lower than 1 or higher than 10. The plain horizontal red line shows the population size (763). White circles show the corresponding data from the influenza A (H1N1) outbreak at a British boarding school (these data are just illustrative here and do not influence the other quantities).

can make sense to run as many chains as there are processor cores and reduce the number of iterations per chain accordingly (down to a certain point). For complex models it can be necessary to resort to high-performance computing clusters.

4.6 Criticizing the inference

After fitting the model, we check whether the inference is reliable. In this setting, this means validating that the samples are unbiased and that our posterior estimates, based on these samples, are accurate.

4.6.1 `Stan`'s diagnostics

`Stan` provides a host of information to evaluate whether the inference is reliable. During sampling, `Stan` issues warnings when it encounters a numerical error: for example, failure to solve the ODEs or to evaluate the log joint distribution, $\log p(\mathcal{Y}, \theta)$. A large number of such failures is indicative of numerical instability, which can lead to various kinds of problems: slow computation or inability to generate samples in certain regions of the parameter space. These are often caused by coding mistakes, and the first step towards a solution is debugging the `Stan` code. Fortunately, with the example at hand, we observe no such warning messages.

After the sampling is finished, `Stan` will run a set of basic diagnostics and issue a warning if it finds divergent transitions, saturated maximum tree depth or low energy[‡]. Of these, the most serious is the presence of divergent transitions as it really challenges the validity of the inference, and requires revisiting the model or tweaking the inference settings. Exceeding maximum tree depth is a concern about sampling efficiency rather than validity. A low energy warning can be more problematic, as it suggests that the Markov chains did not explore the posterior distribution efficiently. Running the inference again with more iterations is often sufficient to solve this issue. More details about `Stan`'s warnings and solutions are available in [35].

The summary table provides several quantities to troubleshoot the inference:

```
              mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
beta          1.73       0 0.05 1.63 1.70 1.73 1.77  1.84  2544    1
gamma         0.54       0 0.05 0.45 0.51 0.54 0.57  0.63  2275    1
phi_inv       0.14       0 0.07 0.04 0.08 0.12 0.17  0.33  2271    1
```

The $\widehat{R}$ statistics estimates the ratio between the overall variance and the within chain variance: if $\widehat{R}1$, then the Markov chains are not mixing and at least one of the chains is producing biased samples. To measure how

[‡]These diagnostics can also be accessed in `rstan` with the `check_hmc_diagnostics()` function.

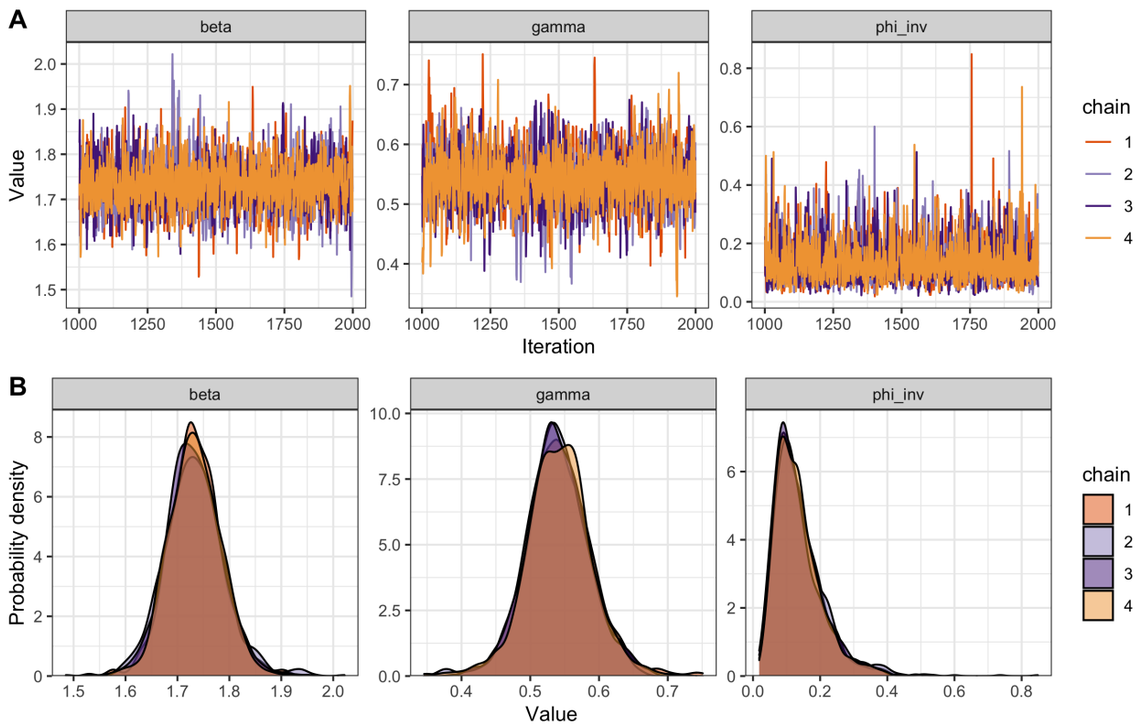

Figure 6: (A) Trace plot showing the value of each chain at each iteration (excluding warm-up) and (B) marginal posterior densities for the transmission rate (`beta` or β), the recovery rate (`gamma` or γ) and the inverse dispersion parameter (`phi_inv` or $1/\phi$), separately for each of the four Markov chains.

"informative" the samples are, we may use the *effective sample size*, n_{eff}, which is typically smaller than the total number of sampling iterations. Samples from an MCMC procedure tend to be correlated, meaning there is redundant information in the samples which makes the posterior estimates less precise. Conceptually, the variance of our Monte Carlo estimates corresponds to the variance we would expect if we used n_{eff} independent samples. Stan's procedure for computing $\widehat{R}$ and n_{eff} is based on recent improvements made to both estimates[36].

Here we note that $\widehat{R}$ is close to 1 (¡ 1.01) and that `n_eff` is large, which makes us confident we can rely on the inference. Conversely, large $\widehat{R}$ and low `n_eff` would indicate that the Markov chains are not cohesively exploring the parameter space and, in turn, that our estimates of the posterior mean and quantiles are unreliable. Having no `Stan` warning is a good, if imperfect, indication that the inference is reliable. We can furthermore plot the "trace" (Figure 6A) and the marginal posterior densities (Figure 6B) of each chain to confirm that the Markov chains mix well and are in agreement with one another.

When the diagnostics reveal a problem with our inference, we must consider several sources of error. A common, if trivial, problem is coding errors. If we are confident in our implementation, we may inspect our model specification. Often times, reparameterizing the model[23] or using stronger priors, when the requisite information is available, improves the interaction between HMC and the model, leading to better mixing of the chains and moreover better inference. Sometimes the model at hand, i.e. the likelihood and the prior, is the model we need to fit and we must entertain the possibility that our inference engine is not suited for the problem at hand. In this case, we may consider changing the tuning parameters of HMC or all together adopt a new inference scheme, e.g. marginalization[37, 38, 39], variational inference[40], or adaptive Metropolis and Gibbs[41], to only name a few popular approaches. Indeed HMC is not a one-size-fits-all solution and, depending on the context, other techniques can offer better performance. The principles of the Bayesian workflow apply to other inference strategies, although we should be mindful that not all algorithms are amiable to diagnostics and that some techniques can require extensive revision whenever we change model (or even be restricted to a narrow menu of models).

4.6.2 Criticize the inference with simulated data

While there exist many theoretical guarantees for MCMC algorithms, modelers should realize that these rely on a set of assumptions which are not always easy to verify and that many of these guarantees are *asymptotic*. This means they are proven in the limit where we have an infinite number of samples from the posterior distribution. A very nice, if advanced, review on the subject can be found in [42]. As practitioners, we must contend with finite computational resources and assumptions which may or may not hold. The diagnostics we reviewed earlier – e.g. $\widehat{R}$ or effective sample sizes – provide *necessary* conditions for the MCMC sampler to work but not *sufficient* ones. Nevertheless, they are potent tools for diagnosing shortcomings in our inference. This subsection provides further such tools, from both a rigorous and a pragmatic perspective.

Fitting the model to simulated data is, if done properly, an effective way to test whether our inference algorithm is reliable. If we cannot successfully fit the model in a controlled setting, it is unlikely we can do so with real data. This of course raises the question of what is meant by "successfully fitting" the model. In a Bayesian setting, this means our inference procedure accurately estimates various characteristics of the posterior distribution, such as the posterior mean, variance, covariance, and quantiles.

A powerful method to check the accuracy of our Bayesian inference is *simulation-based calibration* (SBC)

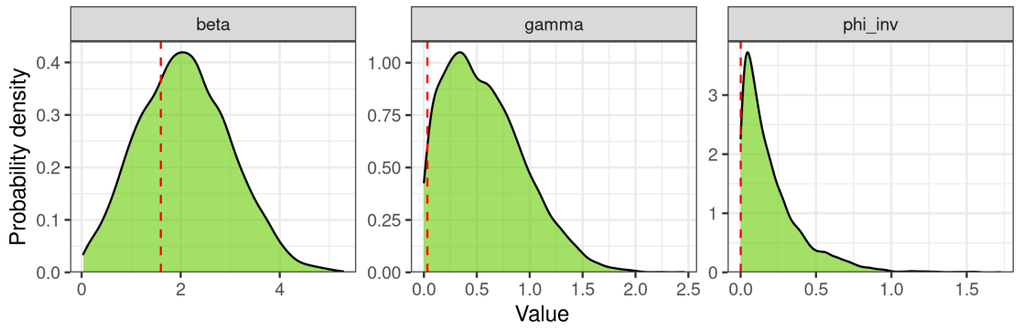

Figure 7: Marginal posterior densities for the transmission rate (`beta` or β), the recovery rate (`gamma` or γ) and the inverse dispersion parameter (`phi_inv` or $1/\phi$) obtained when fitting the model to simulated data. The red dashed lines show the fixed parameter values used for simulating the data.

[43]. SBC exploits a very nice consistency result. The intuition is the following: if we draw several sets of parameters from our prior distribution

$$\theta_1, ..., \theta_2 \sim p(\theta)$$

and for each set of parameters θ_i simulate a data set $\mathcal{Y}^i$, we can by fitting the model multiple times recover the prior distribution from the estimated posteriors. This technique is a bit beyond the scope of this tutorial, though we vividly encourage the reader to consult the original paper.

For the time being, we focus on a simpler heuristic: fit the model to one simulated data set and check if we recover the correct parameter values. There are serious limitations with this approach: when do we consider that the estimated posterior distribution covers the correct value or how do we know if the variance of the posterior is properly estimated? But the test is useful: in this controlled setting, do the chains converge? Is the computation of the log density numerically stable (e.g. are we using the right ODE integrator)? Do our priors prevent the chains from wandering into absurd regions of the parameter space? These are all questions this simple test can help us tackle.

We take one arbitrary draw from the prior distribution and use it as data to which we fit the model. In this specific case, we used the following randomly-selected values: $\beta = 1.6$, $\gamma = 0.033$ and $1/\phi = 0.0007$. We then plot the estimated posterior distribution (Figure 7) along with the "true" parameter values. The density function covers the true parameters, although it is not always centered on it. The latter is not alarming, especially if the parameter values that were selected lie on the tail of the prior distribution. We could repeat this process a few times to get a better sense of the performance of the model together with the inference algorithm.

4.7 Criticize the fitted model

Now that we trust our inference, we can check the utility of our model. Utility is problem-specific and can include the precise estimation of a parameter or predicting future behaviors. In general, it is good to check if our model, once fitted, produces simulations that are consistent with the observed data. This is the idea behind *posterior predictive checks*.

We sample predictions, $\mathcal{Y}_{\text{pred}}$, using the following sequential procedure:

$$\begin{aligned} \theta_{\text{post}} &\sim p(\theta \mid \mathcal{Y}), \\ \mathcal{Y}_{\text{pred}} &\sim p(\mathcal{Y} \mid \theta_{\text{post}}). \end{aligned}$$

This is identical to prior predictive checks, except that we are now sampling parameters from the posterior distribution, rather than from the prior. We use these samples to construct a fitted curve for students in bed, together with a measure of uncertainty (e.g. the 95% prediction interval, meaning that observed data points are expected to fall outside of the interval once every twenty times). The posterior predictive check represented in Figure 8A allows us to confirm that the model fits the data reasonably well. We can also do separate posterior predictive checks for each Markov chain.

At this point, we can confidently move forward with the utility of the model, for instance conclude that the basic reproduction number of this outbreak can be estimated to 3.2 with a 95% credible interval of 2.7 - 3.9 (respectively from the median, 2.5% quantile and 97.5% quantile of the posterior samples), or that the average recovery time can be estimated to 1.8 days with a 95% credible interval of 1.6 - 2.2 (Figure 8B).

5 Scaling-up ODE-based models

Doing MCMC on ODE-based models can be computationally intensive, especially as we scale up the number of observations, parameters, and start using more sophisticated ODEs. If we want to reap the benefits of a full Bayesian inference, we have to pay the computational cost. But while we cannot get away with a free lunch, we can avoid an overpriced one. `Stan` is a flexible language, which means there are worse and better ways of coding things.

This section develops a few principles to make ODE models in `Stan` more scalable, drawing on our experience with an SEIR model of SARS-CoV-2 transmission[3]. The key is to understand the computational cost of each

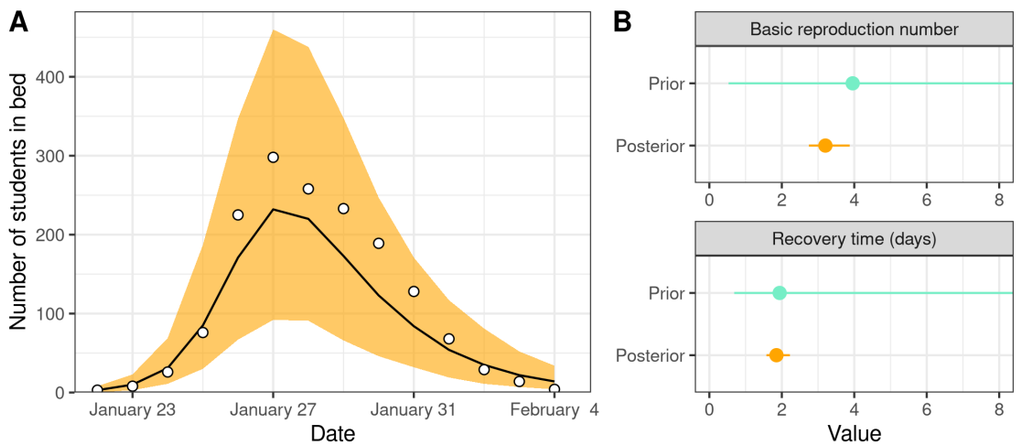

Figure 8: (A) Posterior predictive check of the number of students in bed each day during an influenza A (H1N1) outbreak at a British boarding school. The line shows the median and the orange area the 90% prediction interval. (B) Prior and posterior predictive checks of the basic reproduction number $\mathcal{R}_0$ and of the recovery time (both truncated at 8). The dot shows the median posterior and the line shows the 95% credible interval.

coding block (Section 2.4) and recognize that operations in the `transformed parameters` and `model` blocks dominate the computation. A brief primer on the mechanism of HMC will make this clear. Our goal is then to write our model in such a way that we limit the number of operations in the expensive blocks. Another important consideration is the need to solve and differentiate the ODEs for our transmission model many times. We present ways to make numerical integration cheaper. Anecdotally, implementing the strategies we outline here reduced the runtime of our model from three days to two hours! These techniques will also be much help in our next example in section 6.

5.1 The Computational cost of Stan's coding blocks

`Stan` abstracts the inference away from the modeling but it's worth taking a peek inside the black box. HMC simulates physical trajectories in the parameter space by solving Hamilton's equations of motion – in this instance, we may think of these equations as a convenient reformulation of Newton's law of motion. Unlike random walk algorithms (e.g. Metropolis-Hastings, Gibbs), HMC does not start each iteration with a random step, but rather with a random momentum, given to a fictitious particle. The acceleration of the particle is then driven by the gradient of the log posterior. We solve the equations of motion numerically, using a leapfrog integrator. More details can be found in references[9, 10].

Recall the `model` block specifies the joint distribution $\log p(\mathcal{Y}, \theta)$. We further require the gradient

$$\nabla_\theta \log p(\mathcal{Y}, \theta) = \nabla_\theta \log p(\theta \mid \mathcal{Y}).$$

Fortunately, the user does not need to specify the gradient. Instead, *automatic differentiation* generates the requisite derivatives under the hood using a numerical integrator [21, 44]. At each step the leapfrog integrator takes – that is, multiple times per iteration – we need to *evaluate and differentiate* $\log p(\mathcal{Y}, \theta)$.

This perspective informs how each model block scales (Figure 1). The `data` and `parameters` blocks are used to declare variables. The `transformed data` block is evaluated once per iteration. The `transformed parameters` and `model` blocks are evaluated and differentiated at each integration step, which is multiple times per iteration. The `generated quantities` block is evaluated once per iteration. Hence operations in the `parameters` and `transformed parameters` blocks dominate the computational cost and should only entail operations that depend on θ and are required to compute $\log p(\mathcal{Y}, \theta)$.

In the previous example, solving the ODEs at the observed times is required to compute the likelihood and occurs in the `transformed parameters` block. On the other hand, the computation of $\mathcal{R}_0$ is relegated to `generated quantities`. Even though we want samples from $p(\mathcal{R}_0 \mid \mathcal{Y})$ and $\mathcal{R}_0$ depends on the model parameters, $\mathcal{R}_0$ does not contribute to $\log p(\mathcal{Y}, \theta)$.

5.2 Reducing the cost of propagating derivatives through the ODE solution

To obtain the requisite gradient, we must propagate derivatives through the ODE solution. This differentiation process can become very expensive: understanding its mechanism can help us avoid certain computational pitfalls.

Our ODE is defined by

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y, t, \vartheta, x).$$

Here, ϑ contains inputs to f that depend on the model parameters, θ, while x contains inputs which do *not* depend on θ and therefore remain fixed as the Markov chain moves through the parameter space. Note that in general, $\vartheta \neq \theta$. In the SIR model for example, $\vartheta = \{\beta, \gamma\}$, while $\theta = \{\beta, \gamma, 1/\phi\}$. To define the integral, we additionally specify an initial time t_0, times of integration t_s, and an initial condition y_0, all of which can vary with θ. Hence when propagating derivatives to compute the gradient of the log joint density, we need to worry

about how the solution varies with respect to ϑ and potentially y_0[§]. We call these elements *varying parameters* and denote K the number of such elements. Furthermore, let N be the number of *states*, that is the length of y or the number of compartments in a SIR-type model.

In `Stan`, we propagate derivatives by solving a coupled system of ODEs. The intuition is the following. Suppose we want to compute

$$\frac{\mathrm{d}y}{\mathrm{d}\vartheta}.$$

We do not have an analytical expression for y, so a direct application of automatic differentiation is not feasible. But we can, assuming the requisite derivatives exist, compute

$$\frac{\mathrm{d}f}{\mathrm{d}\vartheta} = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\mathrm{d}y}{\mathrm{d}\vartheta}$$

and then integrate this quantity numerically. The end result is that, instead of only solving for the N original states, we solve an $N + NK$ system to both evaluate and differentiate y.[¶] Evidently, solving and differentiating the ODE is much more expensive than only solving it! We this in mind, our goal should be clear: limit both N and K as much as possible.

The number of states N in SIR-type models is simply the number of compartments and not much can be done to reduce this number. Limiting K is first and foremost a matter of book-keeping. Recall the function signature of `Stan`'s numerical integrator:

```
real[,] y = integrate_ode_rk45 (function f, real[] y0, real t0, real[] ts,
                real[] theta, real[] x_r, int[] x_i);
```

For every element in ϑ, we add an additional N states to solve for. Hence, components which do not depend on θ should be passed through `x_r` and `x_i`[‖].

Suppose our initial condition, y_0, are varying parameters, i.e. depend on the model parameters. It is not uncommon for some of the elements in y_0 to not depend on θ. For example, in a compartment model, the initial condition for the I compartment may depend on model parameters, while it is set to 0 for the R compartment. More generally, y_0 may only depend on $k < N$ parameters. The straightforward approach is to pass y_0 as a vector of parameters. `Stan` interprets this as N additional varying parameters, which means the number of ODE we solve increases by N^2. This is overpriced lunch!

A better, if more intricate, approach is to solve the ODEs for deviations from the baseline and split y_0 between ϑ and `x_r`. Concretely, let

$$z = y - y_0.$$

The initial condition for z is then $\mathbf{0}$, an N-vector of 0's and a fixed quantity. Now,

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{\mathrm{d}y}{\mathrm{d}t} = f(z + y_0, t, \vartheta, x) = \widetilde{f}(z, t, \widetilde{\vartheta}, \widetilde{x})$$

where $\widetilde{f}$ is the same map as f, but applied to $z + y_0$ instead of z; $\widetilde{\vartheta}$ contains ϑ and the elements of y_0 that depend on θ; and $\widetilde{x}$ contains x and the elements of y_0 that are fixed. With this implementation, K is kept to a minimum. We recover the original y simply by adding y_0 to z. In the SEIR model by [**?**, Hauser et al, 2020]]riou2020covid, we have 58 initial conditions but together these depend on a single parameter. ϑ itself contains 4 elements. Reparameterizing the ODE means we go from $K = 62$ to $K = 5$, that is from solving 3596 coupled ODEs to only solving 290.

5.3 Picking the right ODE integrator

The task of solving and differentiating an ODE boils down to integrating an augmented ODE system. The majority of the time, we deal with nonlinear ODEs with no analytical solution and must resort to numerical integrators. Hence to ensure reasonable performance, it is crucial to pick the right integrator and tune it properly.

Conceptually, numerical integrators perform a linear approximation of the solution between time points t and $t + \delta t$, using the tangent, f. The step size δt controls the usual trade-off between accuracy and speed, with a smaller δt leading to more accurate results but slower computation. The step size is adaptively computed by the integrator in order to control the target approximation error. When calling an integrator in `Stan`, the user may specify the target error via

1. the *absolute tolerance*, which sets the upper bound for the absolute error in a solution,

2. the *relative tolerance*, which sets the upper bound for the error relative to the solution.

[§] While t_0 and t_s may depend on model parameters, we here assume they do not to avoid some minute technicalities and simplify our discussion.

[¶] Strictly speaking, we do not need to explicitly compute $\mathrm{d}y/\mathrm{d}\vartheta$ to propagate derivatives; this is an important, if somewhat counter-intuitive, result of automatic differentiation, and motivates a so-called *adjoint method*, which only requires solving $2N + K$ ODE states, albeit incurring additional overhead cost. For a deeper discussion on the topic, we recommend [**?**, Hindmarsh and Serban, 2020]]hindmarsh2020ode.

[‖] Stan now offers a variadic signature, which is more flexible and automatically recognizes if an argument is parameter dependent or not[45].

Iteration	Model	Data	Stan diagnostics	Other issues	Interpretation / Comments	Proposed improvement
#1	Basic SIR model	Reported cases	$\hat{R} >> 1$, max_treedepth exceeded	Far-off predictions	Misspecified model	Add a reporting rate parameter p_r
#2	SIR + underreporting	Reported cases	None	Skewed prediction	Misspecified model	Add incubation rate
#3	SEIR + underreporting + incubation rate	Reported cases	$\hat{R} > 1.01$	1 out of 4 chains with skewed predictions and unrealistic incubation time	Degeneracy due to weak priors	More informative prior on incubation time
#4	SEIR + underreporting + varying initial infections + informative prior on incubation time	Reported cases	$\hat{R} > 1.01$	Unrealistic values of p_r for some chains	The model can only make infections decrease if p_r is very low	Model control measures
#5	SEIR + underreporting + varying initial infections + model control measures	Reported cases	A few divergences	p_r is unrealistic, degenerate values for logistic parameters	Degeneracy between the two ways to make infections decrease: immunity and control measures	Inform p_r by including serological data
#6	SEIR + underreporting + varying initial infections + model control measures	Reported cases + serology	max_treedepth exceeded for most iterations, a few divergences	None	Unclear	Increase the maximum tree depth
#7	SEIR + underreporting + varying initial infections + model control measures + increased max tree depth	Reported cases + serology	None	None	The model seems decent!	None

Table 1: Summary of the Bayesian workflow used for iteratively building a model of SARS-CoV-2 transmission in Switzerland.

Lower tolerance typically induces smaller step sizes. Unless the user has a strong grasp on how these tuning parameters may affect the inference, we recommend using `Stan`'s default. Another important tuning parameter is the *maximum number of steps*, after which the integrator gives up on solving the ODE. In a Bayesian context, it can be useful to end the integrator early, especially when the Markov chain wanders into odd regions of the parameter space, that make the ODE difficult to solve. Such a wandering can occur during the warmup phase of HMC, before the chains converge to the relevant region of the parameter space.

`Stan` supports two ODE integrators[45]: a Runge-Kutta (RK45) method for non-stiff systems with `integrate_ode_rk` and a backward differentiation (BDF) algorithm for stiff systems with `integrate_ode_bdf`. There is no formal definition of stiffness but the general idea is that the phenomenon occurs when the step size, δt, of the integrator needs to be extremely small – smaller than what is needed to achieve the required accuracy – in order to make the integrator stable. Stiffness can arise when the scale of the solution changes varies largely as a function of t. The RK45 integrator is typically faster, so we recommend it as a starting point. If however the system is stiff, the RK45 integrator will be slow and numerically unstable and in this case, the BDF integrator is preferable.

Users should therefore be prepared to adjust their solver. This can mean switching from RK45 to BDF, or adjusting the tuning parameters of the integrator. When an integrator fails to solve an ODE, `Stan` issues a warning message and rejects the iteration being computed. An excessive number of such failures can indicate the integrator needs to be adjusted.

For certain problems, knowing ahead of time if a system is stiff may not be obvious. What is even less obvious is whether a coupled system is stiff. And what is yet again less obvious is whether a system is stiff or nonstiff across the range of parameters the Markov chain explores, both during the warm-up and the sampling phases. As said, the chain can, during the warmup phase, land in extreme regions of the parameter space and the ODE can become tremendously difficult to solve, leading to slow computation and numerical instability. In our experience, this unfortunate behavior is often difficult to avoid. Using more informative priors, when such information is available, can help the chains converge faster to "sensible" regions of the parameter space. Carefully picking initial values for the Markov chain can also be helpful.

6 Application to SARS-CoV-2 transmission in Switzerland

In section 4, we demonstrated how to use `Stan` to build a basic disease transmission model on the example of influenza data. We now show how to build a more complex model of SARS-CoV-2 transmission in Switzerland. We start with the SIR model from section 4, and then gradually build on it by iteratively increasing the complexity of the model and keeping in mind the principles of the Bayesian workflow described in section 3. At each step, we aim to identify and eliminate a specific shortcoming. All obstacles can be roughly classified as inference issues or modeling issues. Inference issues can be, in most cases, diagnosed by `Stan`. They include errors in code, biases in MCMC, or unidentifiable parameters. Modeling issues include misspecified or unrealistic models that pass the inference diagnostics but do not describe the data well or lead to impossible parameter values. These can be detected by prior and posterior predictive checks. All through, we keep the *folk's theorem* [8] in mind, according to which an inference problem is often due to a modeling problem. For example, poorly specified priors can put probability mass in numerically unstable regions of the parameter space and frustrate our inference algorithms. Table 1 presents every iteration of our model. The complete analysis, including the `Stan` code for the listed models, is available in a complementary notebook[11]. Below we only present a broad outline of the iterative modeling process.

6.1 Data

The main dataset used in this section is the daily number of reported cases of SARS-CoV-2 infection in Switzerland at the national level during the first epidemic wave, from February to June 2020 (Figure 9). At

a later modeling stage, we include serological antibody survey data from Geneva, available for dates between April 6th and May 9th 2020[46].

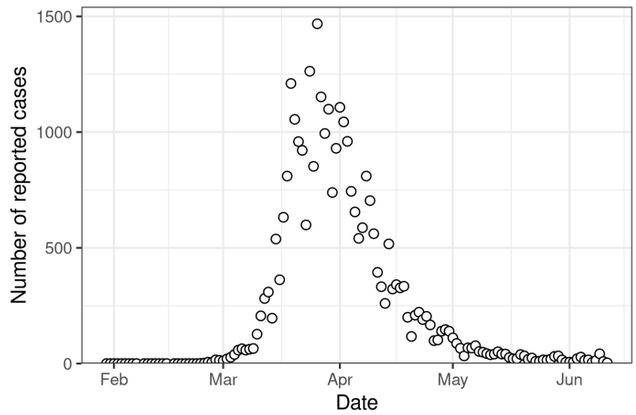

Figure 9: Daily number of reported cases of SARS-CoV-2 infection in Switzerland between February and June, 2020.

6.2 First attempt

As a first step, we directly try to fit the SIR model from section 4 to this new data (model iteration #1 in Table 1). We do not expect this simple model to perform well, but rather aim to create a baseline. We first need to account for one significant difference between influenza and SARS-CoV-2 datasets: the former represents prevalence (the number of students in bed being interpreted as a proxy for currently infected individuals), and the latter represents incidence (the number of new cases on a given day). To adjust for this discrepancy, we compute the incidence $\Delta I(t)$ from the compartments of the SIR model as the number of individuals entering the I compartment during day t. The new sampling distribution is:

$$p(\mathcal{Y} \mid \theta) = \text{Negative-binomial}(\mathcal{Y} \mid \Delta I(t), \phi)$$

where $\mathcal{Y}$ now denotes data on reported cases.

Several diagnostics indicate that the inference from this first model fit should not be trusted: `Stan` issues a warning about divergent transitions, $\hat{R}$ is far larger than 1, and the effective sample size is very small. We can also see that the Markov chains do not mix (Figure 10A), and a posterior predictive check further shows that the model is, across all chains, unable to fit the data (Figure 10B). It is not uncommon for problems to come in bulk. This is actually a feature of `Stan`: when it fails, it fails loudly.

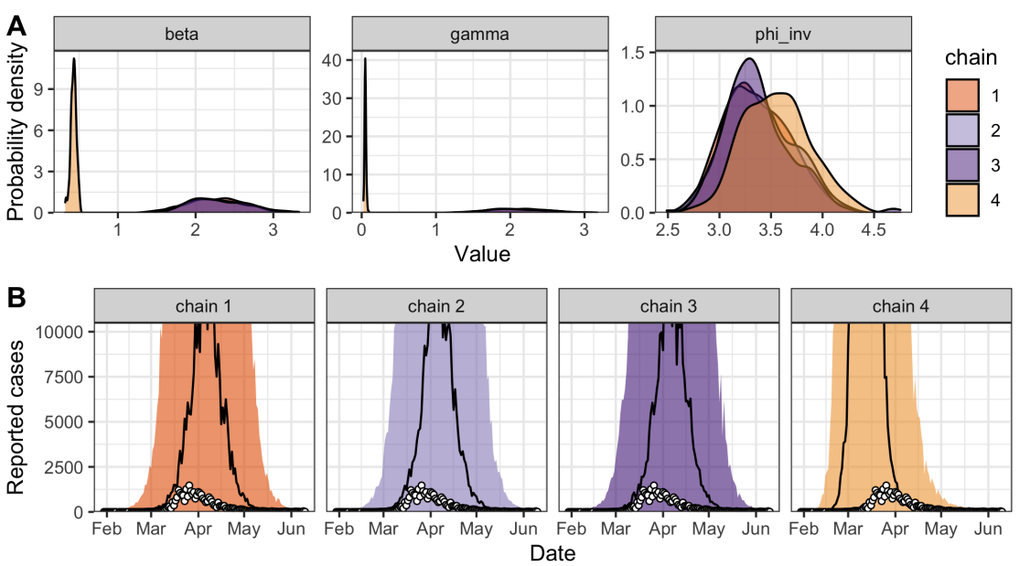

Figure 10: (A) Posterior distributions of the model parameters (the transmission rate β, the recovery rate γ and the inverse dispersion parameter $1/\phi$) and (B) chain-by-chain posterior predictive check of the number of reported cases for the simple SIR model (model iteration #1) applied to data on the SARS-CoV-2 epidemic in Switzerland (white circles).

6.3 Model improvement

Drawing from domain knowledge, we take several steps to improve the model.

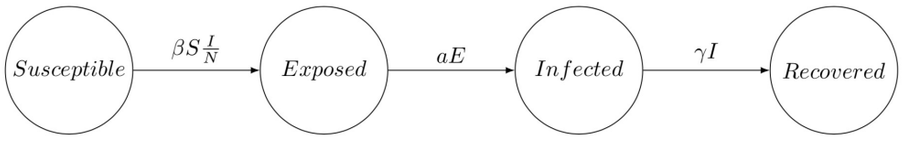

Figure 11: Diagram of a SEIR model.

6.3.1 From SIR to SEIR with underreporting

First, we add a reporting rate parameter, $p_r \in (0, 1)$, to account for the underreporting of cases (model iteration #2). Indeed, contrary to the controlled environment of the boarding school where every student can be monitored, reported cases are an incomplete measurement of the true incidence of SARS-CoV-2. In order to be reported as cases, individuals infected with SARS-CoV-2 have to get tested, which is not proposed to everyone and much more likely to be proposed to individuals with severe symptoms or a perceived risk of severe outcome[47]. Also, the test has to be positive given that the individual tested is infected, which depend on the test sensitivity. We combine these elements in a single parameter, and select a weakly-informative prior, $p(p_r) = \text{Beta}(1, 2)$. We still ignore the fact that reporting may take a few days, and assume that infected individuals appear as reported when they become infectious (as then enter compartment I).

Second, we add an *Exposed* compartment E to account for individuals who have been exposed to the virus but are not yet infectious. This leads to an SEIR model (model iteration #3) that features another additional parameter: the *incubation rate* a, defined as the inverse of the average incubation time. We select a weakly-informative prior for this new parameter, $p(a) = \text{Normal}^+(0.4, 0.5)$, which encodes the belief that exposed individuals become infectious after a period that lies between 0.5 and 30 days.

6.3.2 Limitations to the SEIR model with underreporting

Unfortunately, new obstacles appear when we try to fit this more complex model. `Stan` informs us that the chains are not mixing. To understand the issue, we try several diagnostics. Trace plots (Figure 12A) show that chains converge to two different modes when exploring the posterior distributions of the incubation rate a and the reporting rate p_r. A chain-by-chain posterior predictive check (Figure 12B) shows that all four chains give better predictions than the previous models, but only chains 2 and 4 produce satisfying results. At this point, it is useful to look at the parameter spaces explored by different chains. In this case, we observe that chains 1 and 3 explore regions where the incubation rate a is close to 2, so that the incubation time $1/a$ is very short, about 0.5 days. However, we know from the literature that the average incubation time has been estimated around 5-6 days[48]. This inconsistency suggests the incubation time cannot be inferred from the data alone and motivates incorporating more expert knowledge in the model in the form of an informative prior. We reparametrize the incubation rate as its inverse, and set an informative prior on the incubation period centered around 6 days: $p(1/a) = \text{Normal}^+(6, 1)$ (model iteration #4). Unfortunately, even after this correction, the chains still do not mix (not shown).

We also notice that another parameter does not agree with domain knowledge: the probability that an individual infected with SARS-CoV-2 is reported as a case is very low for most chains, around 0.3-0.8%. Such low values would imply that the entire population of Switzerland had been infected by July (30,994 reported cases over the period divided by 0.5% gives more than 10 millions infected). However, local serological studies [46] have shown that the cumulative number of infections in Geneva was not far from 10% of the population by May 2020. This behavior of the current model can be explained. In this configuration of the model, transmission can only decrease due to acquired immunity (i.e. by lack of susceptible individuals). As the data show a decrease in reported cases, the model has to conclude that a large part of the population is immune at this point, and thus that the number of cases reported until then represents a very small proportion of the true number of infections. Given the data from the serological studies, the decrease in transmission must be due to something other than immunity. Indeed on March 17, the Swiss government implemented lockdown measures to stop the spread of the disease. Our model should therefore account for this effect.

6.4 Modeling control measures

We model the decrease in transmission after March 17 and the implementation of lockdown measures ** using a *forcing function* with a logistic shape:

$$\beta^*(t) = f(t)\beta,$$

with

$$f(t) = \eta + (1 - \eta)\frac{1}{1 + \exp\left(\xi(t - t_1 - \nu)\right)},$$

where η is the decrease of transmission when control measures are fully in place, ξ is the slope of the decrease, and ν is the delay (after the date of introduction of control measures, t_1) until the measures are 50% effective (Figure 14C). We add weakly-informative priors on the three parameters: $p(\eta) = \text{Beta}(2.5, 4)$ which means that we expect lockdown measures to reduce transmission, but not all the way to zero; $p(\xi) = \text{Uniform}(0.5, 1.5)$,

**Note that we don't model explicitely the behavior changes from the population independent of the lockdown, so $f(t)$ could be interpreted as the combined effect of lockdown and independent behavior changes.

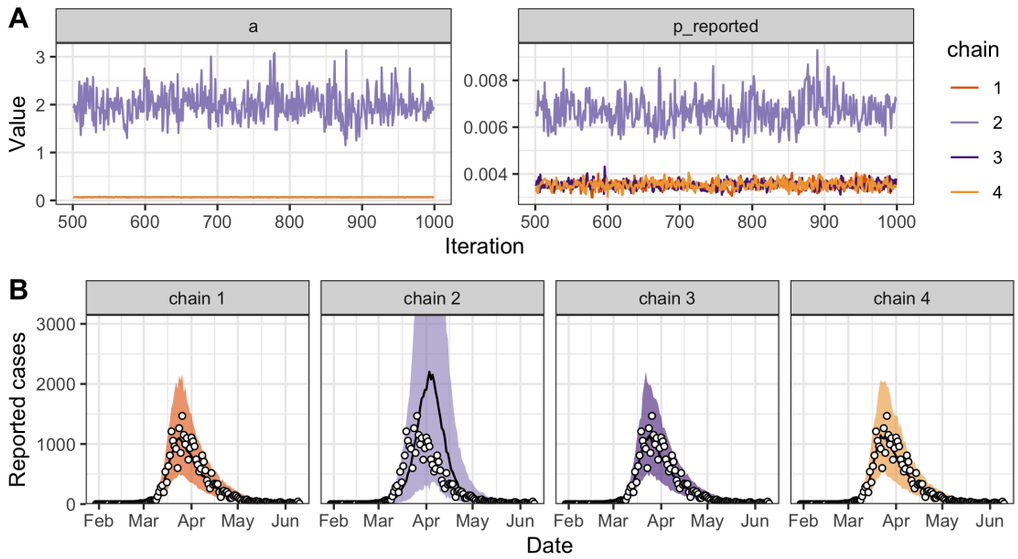

Figure 12: (A) Trace plot for two of the model parameters (the incubation rate a and the reporting rate p_r – the other parameters are not shown) and (B) posterior predictive check of the number of reported cases for the SEIR model with underreporting (model iteration #3) applied to data on the SARS-CoV-2 epidemic in Switzerland (white circles).

which implies that the slope has to be positive but not too steep; and $p(\nu) = \text{Exponential}(0.2)$ which means that the delay before lockdown reaches half of its total efficiency should be between 0 and 20 days.

With this new formulation (model iteration #5), the model should be able to describe the dynamics of SARS-CoV-2 transmission during the first wave of the pandemic. However, our work is not finished yet, as the sampler appears to have trouble exploring the posterior distributions: `Stan` issues a warning about divergent transitions, which indicate the chains may not be exhaustively exploring the parameter space and are thus producing biased samples. The other diagnostics are concerning: $\widehat{R}$ values go up to 1.19 and are above 1.01 for 6 of 8 model parameters, effective sample sizes are low, below 50 for 3 parameters. Another concern is that the probability of a case being reported, p_r remains very low, between 0.3 and 0.4%. To understand what may be happening, we resort to another, more advanced diagnostic tool: the pairs plot (Figure 13). The pairs plot shows samples across pairs of parameters and is useful to examine the geometry of the posterior density. In this pair plot, divergent transitions, shown in red, do not concentrate in a particular zone of parameter space that could point towards a specific issue[49]. We observe that the posteriors are not well-defined bivariate bell curves but show some strong correlation (e.g., between β and $gamma$) and some irregular shapes (e.g. the spike between a and ν). These points can be indicative of identifiability issues: several combinations of parameter values may result in the same observation. Furthermore, we note that the marginal posterior distributions of η, ν and ξ, the three parameters controlling the effect of lockdown measures, are very similar to their prior distributions. This posterior behavior's is not surprising: when p_r is that low, transmission can decrease substantially through immunity only, rendering the effect of lockdown measures non-identifiable, or at least degenerate[50]. We however believe that it is highly unlikely for p_r to be very low and we could account for this using a more informative prior for p_r. In this case, we directly include serological survey results into the model, taking advantage of complementary data.

6.5 Fitting data from a serological survey

For simplicity, we only use data from week 5 of the survey. We ignore considerations about test sensitivity and specificity[51], and consider that all recovered individuals will test positive to the serological test. We also make the strong assumption that the results from this survey in Geneva are representative of the whole population of Switzerland. Tests from week 5 have taken place in Geneva from May 4 to May 7, and 83 out of 775 tested individuals were found to have antibodies against SARS-CoV-2. Given these assumptions, the probability of being detected by the survey is the proportion of individuals in the R compartment at the time when the survey was conducted. We expand the sampling distribution defined in section 6.2, multiplying the likelihood of data on reported cases by the likelihood of serological data:

$$p(\mathcal{Y}' \mid \theta) = \text{Binomial}\left(\mathcal{Y}' \mid \frac{R(t_{\mathcal{Y}})}{N}, n_{\mathcal{Y}}\right)$$

where $\mathcal{Y}'$ is the number of positive tests in the survey sample, $t_{\mathcal{Y}}$ is the time of the survey and $n_{\mathcal{Y}}$ is the sample size. The implementation in code is straightforward (model iteration #6).

Running this new iteration of the model, `Stan` issues a warning that the tree depth is often exceeded. This means that the sampler had to choose a step size small enough to explore some part of the posterior, but that this step size is too small for exploring another part of the posterior efficiently, slowing down the sampling process. This is not too worrying, given that the other diagnostics are good. We can increase the maximum tree depth and try again (model iteration #7). Finally this time, no warning is issued, all diagnostics are good, and the prior and posterior predictive checks confirm the reliability of the inference and the relative adequacy of the model (Figure 14A-B). We can go forward with the application of the model and discuss the results – always in

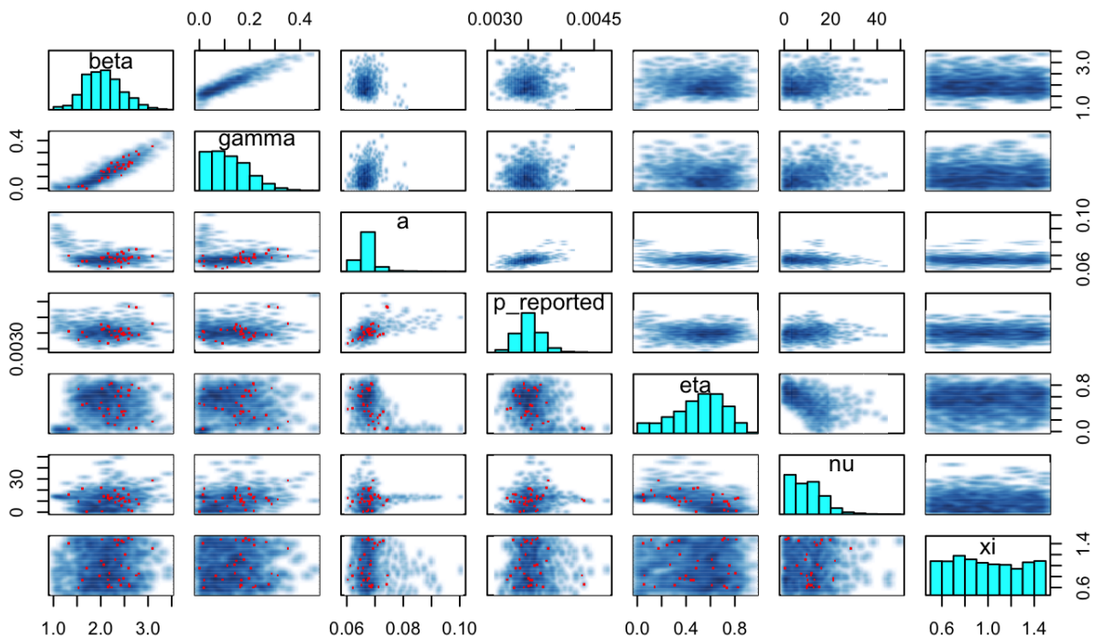

Figure 13: Pairs plot of all model parameters for the model incorporating control measures (model iteration #5).

the context of the model assumptions and priors. For instance, we might want to report that we estimate the basic reproduction number $\mathcal{R}_0$ to 2.7 (95% credible interval 1.9-5.2) and the reporting rate to 3.3% (2.7-4.1). We can also focus on the effects of lockdown measures, interpreting $1-\eta$ as a relative reduction of 73% (53-92) in transmissibility after lockdown measures are fully efficient, or interpreting ν as an implementation delay of 7.5 days (6.2-8.9) before lockdown measures are half-efficient.

We finally obtained a decent model, but note that its aim is only didactic and should not be directly used to inform policy. The first functioning model is not the end of the road, and a lot of features should be considered before claiming to have obtained a good depiction of the Swiss epidemic. For instance, one could account for the sensitivity and specificity of tests when fitting seroprevalence data, improve the inference by including data on testing, hospitalisations and deaths, relax some of the strong assumptions that were made, and stratify by age, location or some other characteristic. This would result in a large number of competing models that can be organised in a *network* depending on what features are included[8]. Model comparison tools become invaluable in this context. Although it is out of the scope of this tutorial, we recommend using *leave-one-out cross-validation*[52], or, even more adapted for time series, *leave-future-out cross-validation* [53].

7 Conclusions

Modeling is an iterative process. We rarely start with a good model. Rather we must build our way to a good model, starting from a baseline, and revising our models as we uncover shortcomings. The Bayesian modeling workflow offers a perspective, which goes beyond fitting a polished model or using inference algorithms in an idealized context; it encompasses failing cases, and techniques to diagnose and learn from these failures. In order to navigate the workflow, we must reason about the modeled phenomenon, our inference algorithms, our computational implementation, and how all these elements interact with one another. This means leveraging domain, statistical, and computational expertise.

We show the advantages that can be obtained from this Bayesian workflow in epidemiology. `Stan` is an adequate tool for building, fitting, and criticizing ODE-based models, as we demonstrate on an influenza model and a more sophisticated SARS-CoV-2 model. For the latter, several ingredients are required to build an adequate model. First and foremost, using an appropriate epidemiological model for a given disease is key. Secondly, in order to improve our estimates of the parameters and overcome issues of identifiability, we need to incorporate information by using well-motivated priors and combining data from multiple sources. Finally, we must tune our inference algorithms in order to accurately probe the posterior distribution. We discover these ingredients by gradually building our model and deploying a broad range of diagnostics. One important practical point is that we need computationally efficient implementations of our code to, in order to not only get reasonably fast inference for the final model, but also to quickly troubleshoot failing models.

As a final thought, we note that the modes of failures for models we develop along the way improve our understanding of the final model. Sharing these early models can therefore make our work more transparent and improve scientific communication.

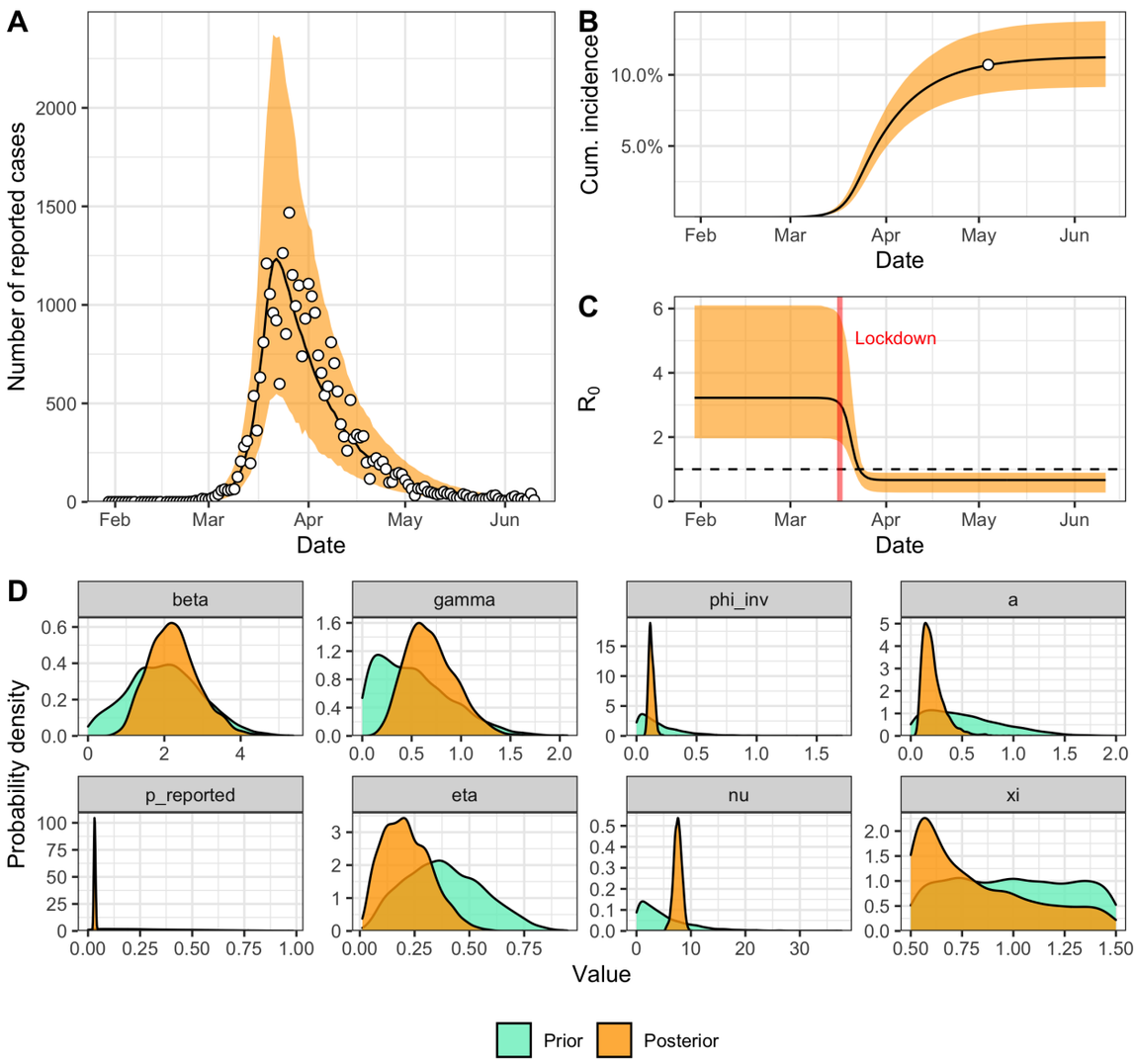

Figure 14: (A) Posterior predictive check of the number of reported cases and (B) of the cumulative incidence for the SEIR model including the effect of control measures and fitted to both reported cases and seroprevalence data (model iteration #7; white circles show data on reported cases in panel A and seroprevalence data in panel B). (C) Posterior distribution of the forcing function f that models the reduction in transmission after the introduction of lockdown measures. (D) Prior and posterior distributions of the parameters of model iteration #7.

Code

The code to run the examples in this article can be found at `https://github.com/charlesm93/disease_transmission_workflow`.

Acknowledgments

We thank Ben Bales and Andrew Gelman for their helpful comments.

Author contributions

LG, ES, CCM and JR conceived this tutorial and wrote the manuscript. LG and ES performed the computations.

Financial disclosure

ES was supported by AstraZeneca postdoc programme. JR is funded by the Swiss National Science Foundation (grant 174281).

Conflict of interest

The authors declare no conflict of interests.

References

[1] Seth Flaxman, Swapnil Mishra, Axel Gandy, H Juliette T Unwin, Thomas A Mellan, Helen Coupland, Charles Whittaker, Harrison Zhu, Tresnia Berah, Jeffrey W Eaton, et al. Estimating the effects of non-pharmaceutical interventions on covid-19 in europe. *Nature*, pages 1–5, 2020.

[2] Henrik Salje, Cécile Tran Kiem, Noémie Lefrancq, Noémie Courtejoie, Paolo Bosetti, Juliette Paireau, Alessio Andronico, Nathanaël Hozé, Jehanne Richet, Claire-Lise Dubost, Yann Le Strat, Justin Lessler, Daniel Levy-Bruhl, Arnaud Fontanet, Lulla Opatowski, Pierre-Yves Boelle, and Simon Cauchemez. Estimating the burden of sars-cov-2 in france. *Science*, 369(6500):208–211, 2020.

[3] Anthony Hauser, Michel J Counotte, Charles C Margossian, Garyfallos Konstantinoudis, Nicola Low, Christian L Althaus, and Julien Riou. Estimation of sars-cov-2 mortality during the early stages of an epidemic: a modeling study in hubei, china, and six regions in europe. *PLoS medicine*, 17(7):e1003189, 2020.

[4] MJ Keeling and L Danon. Mathematical modelling of infectious diseases. *British Medical Bulletin*, 92(1), 2009.

[5] Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017.

[6] Jonah Gabry, Daniel Simpson, Aki Vehtari, Michael Betancourt, and Andrew Gelman. Visualization in bayesian workflow. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 182(2):389–402, 2019.

[7] Michael Betancourt. Towards a principled bayesian workflow. `https://betanalpha.github.io/assets/case_studies/principled_bayesian_workflow.html`.

[8] Andrew Gelman, Aki Vehtari, Daniel Simpson, Charles C Margossian, Bob Carpenter, Yuling Yao, Lauren Kennedy, Jonah Gabry, Paul-Christian Bürkner, and Martin Modrák. Bayesian workflow. *arXiv preprint arXiv:2011.01808*, 2020.

[9] Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

[10] Michael Betancourt. A conceptual introduction to hamiltonian monte carlo. 2018.

[11] Leo Grinsztajn, Elizaveta Semenova, Charles C. Margossian, and Julien Riou. Stan case studies, volume 7 (2020). bayesian workflow for disease transmission modeling in stan. `https://mc-stan.org/users/documentation/case-studies/boarding_school_case_study.html`.

[12] Anastasia Chatzilena, Edwin van Leeuwen, Oliver Ratmann, Marc Baguelin, and Nikolaos Demiris. Contemporary statistical inference for infectious disease models using stan. *Epidemics*, 29:100367, 2019.

[13] Joseph Mihaljevic. Estimating transmission by fitting mechanistic models in stan. 2016.

[14] Bob Carpenter. Stan case studies, volume 5 (2018). predator-Prey Population Dynamics: the Lotka-Volterra model in Stan. https://mc-stan.org/users/documentation/case-studies/lotka-volterra-predator-prey.html.

[15] Sebastian Weber. Solving odes in the wild: Scalable pharmacometrics with stan. *StanCon Helsinki 2018*, 2018.

[16] Charles C Margossian and William R Gillespie. Differential equation based models in stan. In *StanCon 2017*, January 2017.

[17] Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 3rd ed. edition, 2013.

[18] Radford M. Neal. Mcmc using hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*. Chapman & Hall / CRC Press, 2010.

[19] S. Duane, A. D. Kennedy, B. J. Pendleton, , and D. Roweth. Hybrid monte carlo. *Physics Letters B*, 195:216 – 222, 1987.

[20] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18:1 – 43, 2018.

[21] Charles C. Margossian. A review of automatic differentiation and its efficient implementation. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 9, 3 2019.

[22] Stan Development Team. Problematic posterior. `https://mc-stan.org/docs/2_26/stan-users-guide/problematic-posteriors-chapter.html`, 2020.

[23] Stan Development Team. Reparameterization. 2020.

[24] Stan Development Team. Stan interfaces. https://mc-stan.org/users/interfaces/, 2020.

[25] Michael Betancourt. An introduction to stan. https://betanalpha.github.io/assets/case_studies/stan_intro.html.

[26] Stan Development Team. Stan reference manual. https://mc-stan.org/docs/2_26/reference-manual/index.html, 2020.

[27] Stan Development Team. Stan user guide. https://mc-stan.org/docs/2_26/stan-users-guide/index.html, 2020.

[28] Stan Development Team. Stan case studies. https://mc-stan.org/users/documentation/case-studies.html, 2020.

[29] Stan Development Team. Stan tutorials. https://mc-stan.org/users/documentation/tutorials.html, 2020.

[30] George EP Box. Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799, 1976.

[31] David M Blei. Build, compute, critique, repeat: Data analysis with latent variable models. *Annual Review of Statistics and Its Application*, 1, 2014.

[32] Repidemics consortium. outbreaks: a compilation of disease outbreak data. https://www.repidemicsconsortium.org/outbreaks/, 2020.

[33] Stan Development Team. Prior choice recommendations. https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations, 2020.

[34] Ben Bales and Sebastian Weber. Stan case studies, volume 7 (2020). upgrading to the new ode interface. https://mc-stan.org/users/documentation/case-studies/convert_odes.html.

[35] Stan Development Team. Brief guide to stan's warnings. https://mc-stan.org/misc/warnings.html, 2020.

[36] Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner. Rank-Normalization, Folding, and Localization: An Improved $\widehat{R}$ for Assessing Convergence of MCMC. *Bayesian Analysis*, pages 1 – 28, 2021.

[37] Havard Rue, Sara Martino, and Nicolas Chopin. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of Royal Statistics B*, 71:319 – 392, 2009.

[38] Havard Rue, Andrea Riebler, Sigrunn Sorbye, Janine Illian, Daniel Simson, and Finn Lindgren. Bayesian computing with INLA: A review. *Annual Review of Statistics and its Application*, 4:395 – 421, 2017.

[39] Charles Margossian, Aki Vehtari, Daniel Simpson, and Raj Agrawal. Hamiltonian monte carlo using an adjoint-differentiated laplace approximation: Bayesian inference for latent gaussian models and beyond. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 2020.

[40] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, Apr 2017.

[41] Gareth O Roberts and Jeffrey S Rosenthal. Examples of adaptive mcmc. *Bayesian Computation*, 18, 2009.

[42] Gareth O Roberts and Jeffrey S Rosenthal. General state space markov chains and mcmc algorithms. *Probability survey*, 1:20 – 71, 2004.

[43] Sean Talts, Michael Betancourt, Daniel Simpson, Aki Vehtari, and Andrew Gelman. Validating bayesian inference algorithms with simulation-based calibration. *arXiv:1804.06788v1*, 2020.

[44] Andreas Griewank and Andrea Walther. *Evaluating derivatives*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2008.

[45] Stan Development Team. Stan user's guide v2.26, section 13 "ordinary differential equations". https://mc-stan.org/docs/2_26/stan-users-guide/ode-solver-chapter.html, 2021.

[46] Silvia Stringhini, Ania Wisniak, Giovanni Piumatti, Andrew S. Azman, Stephen A. Lauer, Hélène Baysson, David De Ridder, Dusan Petrovic, Stephanie Schrempft, Kailing Marcus, Sabine Yerly, Isabelle Arm Vernez, Olivia Keiser, Samia Hurst, Klara M. Posfay-Barbe, Didier Trono, Didier Pittet, Laurent GÃ©taz, FranÃ§ois Chappuis, Isabella Eckerle, Nicolas Vuilleumier, Benjamin Meyer, Antoine Flahault, Laurent Kaiser, and Idris Guessous. Seroprevalence of anti-SARS-CoV-2 IgG antibodies in Geneva, Switzerland (SEROCoV-POP): a population-based study. *The Lancet*, 396(10247):313–319, 2020. Publisher: Elsevier.

[47] Marc Lipsitch, Christl A Donnelly, Christophe Fraser, Isobel M Blake, Anne Cori, Ilaria Dorigatti, Neil M Ferguson, Tini Garske, Harriet L Mills, Steven Riley, et al. Potential biases in estimating absolute and relative case-fatality risks during outbreaks. *PLoS neglected tropical diseases*, 9(7):e0003846, 2015.

[48] Stephen A Lauer, Kyra H Grantz, Qifang Bi, Forrest K Jones, Qulu Zheng, Hannah R Meredith, Andrew S Azman, Nicholas G Reich, and Justin Lessler. The incubation period of coronavirus disease 2019 (covid-19) from publicly reported confirmed cases: estimation and application. *Annals of internal medicine*, 172(9):577–582, 2020.

[49] Michael Betancourt and Mark Girolami. Hamiltonian Monte Carlo for hierarchical models. *arXiv:1312.0906v1*, 2013.

[50] Michael Betancourt. Identity Crisis. `https://betanalpha.github.io/assets/case_studies/identifiability.html`, 2020.

[51] Andrew Gelman and Bob Carpenter. Bayesian analysis of tests with unknown specificity and sensitivity. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 69(5):1269–1283, 2020.

[52] Aki Vehtari, Andrew Gelman, and Jonah Gabry. Practical bayesian model evaluation using leave-one-out cross-validation and waic. *Statistics and Computing*, 27:1413–1432, 2017.

[53] Paul-Christian Bürkner, Jonah Gabry, and Aki Vehtari. Approximate leave-future-out cross-validation for bayesian time series models. *Journal of Statistical Computation and Simulation*, 90(14):2499–2523, Jun 2020.