Bayesian workflow for disease transmission modeling in Stan

Léo Grinsztajn, Charles Margossian, Julien Riou, Elizaveta Semenova

Berlin Bayesians Meetup
7 December 2021

Preface

- Objective: fit simple transmission models in Stan
- Based on Grinsztajn et al., 2021 (link)
- Prerequisites:
  - basic programming with `R`
  - general understanding of Bayesian inference
- All material is available on github (link)

Outline

Introduction

Models of disease transmission:

- Interpretability: phenomenological, mechanistic
- Scale: population-based, agent-based
- Framework: deterministic, stochastic
- Data-generating mechanisms: incubation, contagion, immunity, vaccination, mobility...

Introduction

Models of disease transmission:

- Interpretability: phenomenological, mechanistic
- Scale: population-based, agent-based
- Framework: deterministic, stochastic
- Data-generating mechanisms: incubation, contagion, immunity, vaccination, mobility...

Mechanistic + population-based + deterministic

→ ordinary differential equations (ODE)-based compartmental model

Introduction

ODE-based compartmental model:

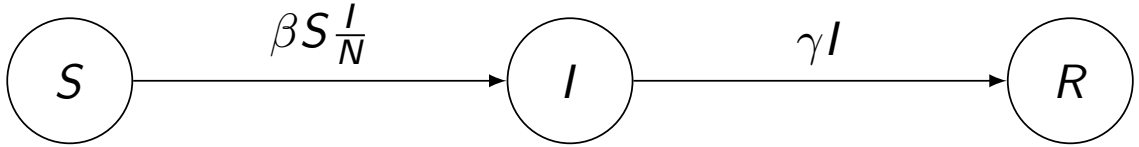
- Divide the population into homogeneous groups (compartments)
- Define the flows between compartments with ODEs
- Define initial conditions (at t_0)
- Solve for the time-dependent volume in each compartment

Introduction

ODE-based compartmental model:

- Divide the population into homogeneous groups (compartments)
- Define the flows between compartments with ODEs
- Define initial conditions (at t_0)
- Solve for the time-dependent volume in each compartment

The susceptible-infectious-recovered (SIR) model:

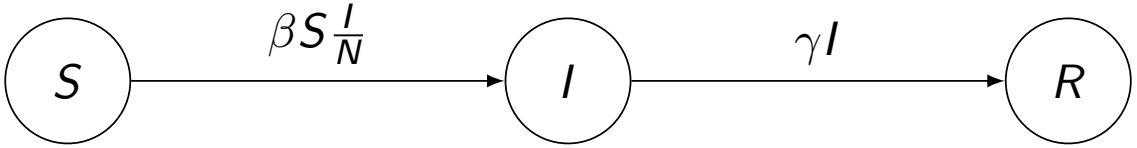


$$\frac{dS}{dt} = -\beta S \frac{I}{N}$$

$$\frac{dI}{dt} = \beta S \frac{I}{N} - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

The SIR model

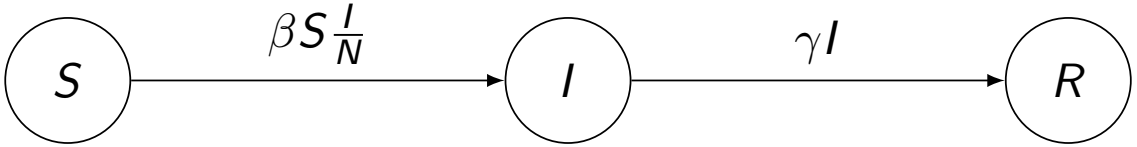


$$\frac{dS}{dt} = -\beta S \frac{I}{N}$$
$$\frac{dI}{dt} = \beta S \frac{I}{N} - \gamma I$$
$$\frac{dR}{dt} = \gamma I$$

Where:

- $S(t)$ is the number of people susceptible to infection
- $I(t)$ is the number of people infected (i.e. the prevalence)
- $R(t)$ is the number of people recovered (lifelong immunity)
- N is the population size ($S(t) + I(t) + R(t) = N$ for any t)
- β is the infectious contact rate (per day per person)
- γ is the recovery rate (1/infectious period)

The SIR model



$$\frac{dS}{dt} = -\beta S \frac{I}{N}$$

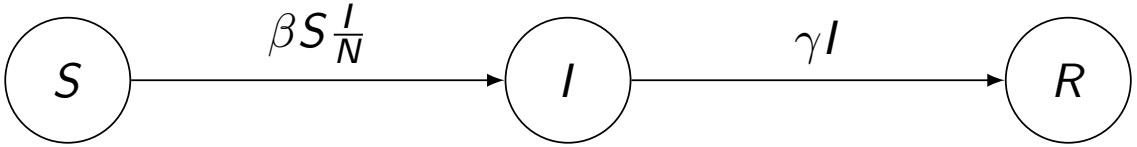
$$\frac{dI}{dt} = \beta S \frac{I}{N} - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

Intuition behind the SIR model:

- $I(t)/N$ is the proportion of infected (and infectious)
- $\beta I(t)/N$ is the daily number of contacts with infectious people
- hence each day, $\beta SI(t)/N$ people become infected (the force of infection)

The SIR model



$$\frac{dS}{dt} = -\beta S \frac{I}{N}$$
$$\frac{dI}{dt} = \beta S \frac{I}{N} - \gamma I$$
$$\frac{dR}{dt} = \gamma I$$

Assumptions behind the SIR model:

- homogeneous mixing
- β and γ constant over time
- all infections are observed
- no incubation, exponentially-distributed recovery
- lifelong immunity
- stable population

Simulate a SIR in R

- load library `deSolve` (and `tidyverse`)
- set compartments and differential equations

```
sir = function(t, x, pars, ...) {
  with(as.list(c(x, pars)), {
    dS = - beta*S*I/(S+I+R)
    dI = beta*S*I/(S+I+R) - gamma*I
    dR = gamma*I
    list(c(dS, dI, dR))
  })
}
```

Simulate a SIR in R

- load library `deSolve` (and `tidyverse`)
- set compartments and differential equations

```
sir = function(t, x, pars, ...) {
  with(as.list(c(x, pars)), {
    dS = - beta*S*I/(S+I+R)
    dI = beta*S*I/(S+I+R) - gamma*I
    dR = gamma*I
    list(c(dS, dI, dR))
  })
}
```

- set (fixed) values for parameters: $\beta = 0.8$; $\gamma = 1/7$

```
pars = c(beta = 0.8, gamma = 1/7)
```

Simulate a SIR in R

- load library `deSolve` (and `tidyverse`)
- set compartments and differential equations

```
sir = function(t, x, pars, ...) {
  with(as.list(c(x, pars)), {
    dS = - beta*S*I/(S+I+R)
    dI = beta*S*I/(S+I+R) - gamma*I
    dR = gamma*I
    list(c(dS, dI, dR))
  })
}
```

- set (fixed) values for parameters: $\beta = 0.8$; $\gamma = 1/7$

```
pars = c(beta = 0.8, gamma = 1/7)
```

- set (fixed) values for initial values

```
N_0 = 100000
I_0 = 50
inits = c(S = N_0 - I_0,
          I = I_0,
          R = 0)
```

Simulate a SIR in R

- solve the ODE system numerically (Runge-Kutta 4th order) to obtain unique solutions for $S(t)$, $I(t)$ and $R(t)$

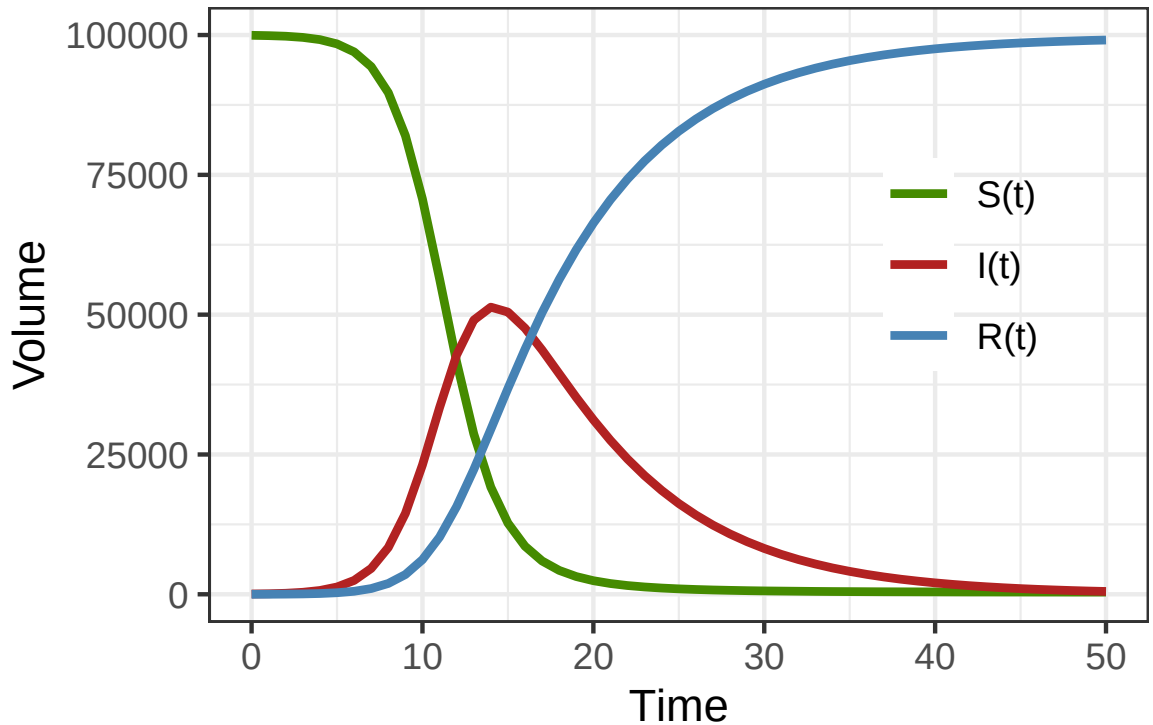
$$f(\beta, \gamma, S_0, I_0, R_0) = \{S(t), I(t), R(t)\}$$

```
times = seq(0,50,by=1)
sim_data = ode(inits, times, sir, pars, method="rk4")
```

```
> tibble(sim_data)
# A tibble: 51 x 1
   sim_data[,"time"] [,"S"]  [,"I"] [,"R"]
               <dbl>  <dbl>   <dbl>  <dbl>
 1                 0 99950     50      0
 2                 1 99894.    96.3   10.1
 3                 2 99785.   186.    29.5
 4                 3 99576.   357.    66.9
 5                 4 99176.   685.   139.
 6                 5 98415.  1308.   276.
 7                 6 96984.  2478.   538.
 8                 7 94350.  4621.  1030.
 9                 8 89692.  8374.  1934.
10                 9 82009. 14457.  3533.
# … with 41 more rows
```

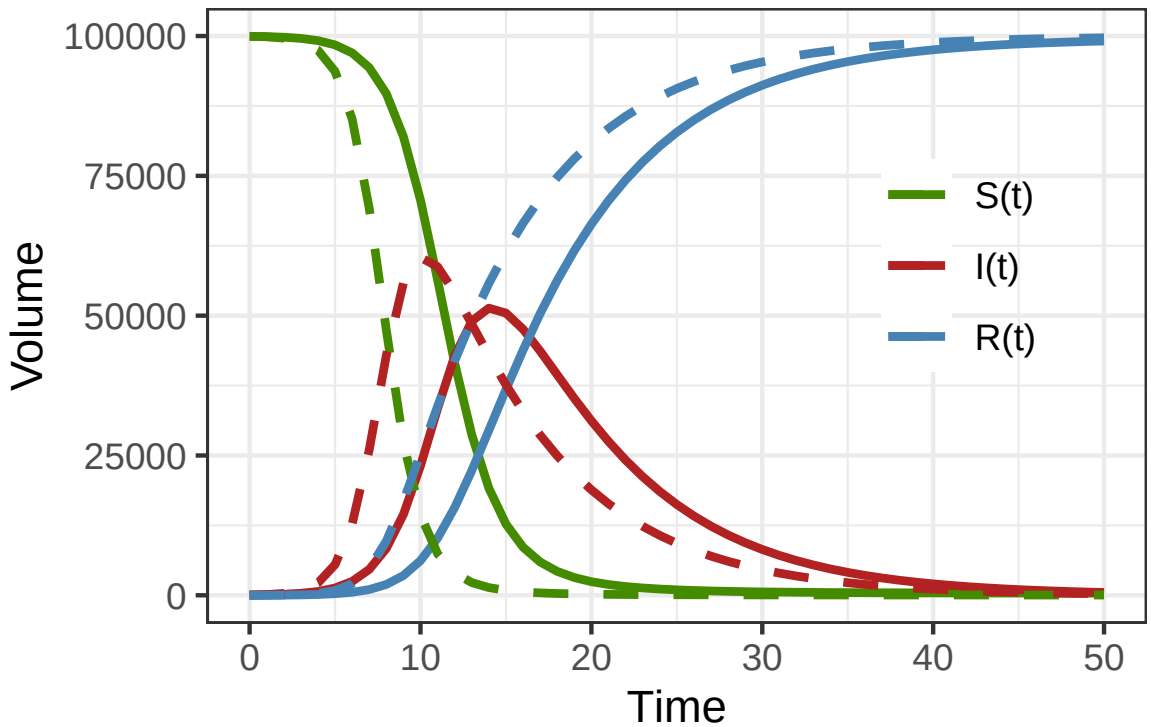
Simulate a SIR in R

with $\beta = 0.8$; $\gamma = 1/7$; $S_0 = 100000 - 50$; $I_0 = 50$ and $R_0 = 0$



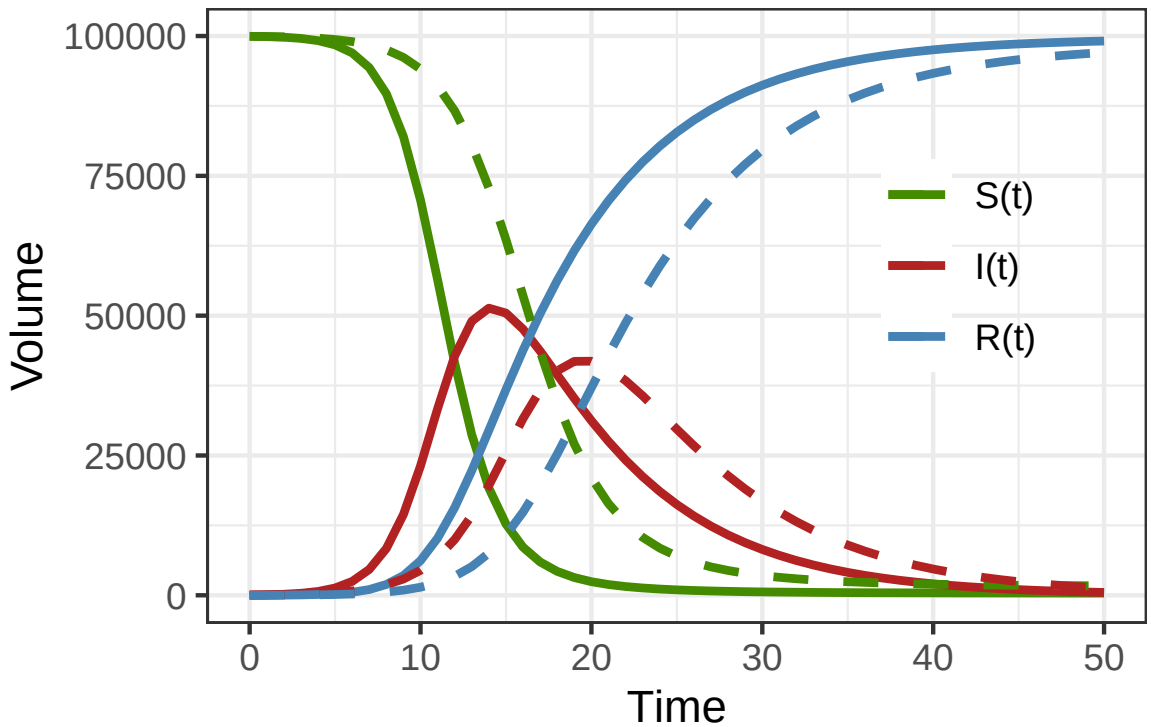
Simulate a SIR in R

with $\beta = 1.1$ instead of 0.8, we get



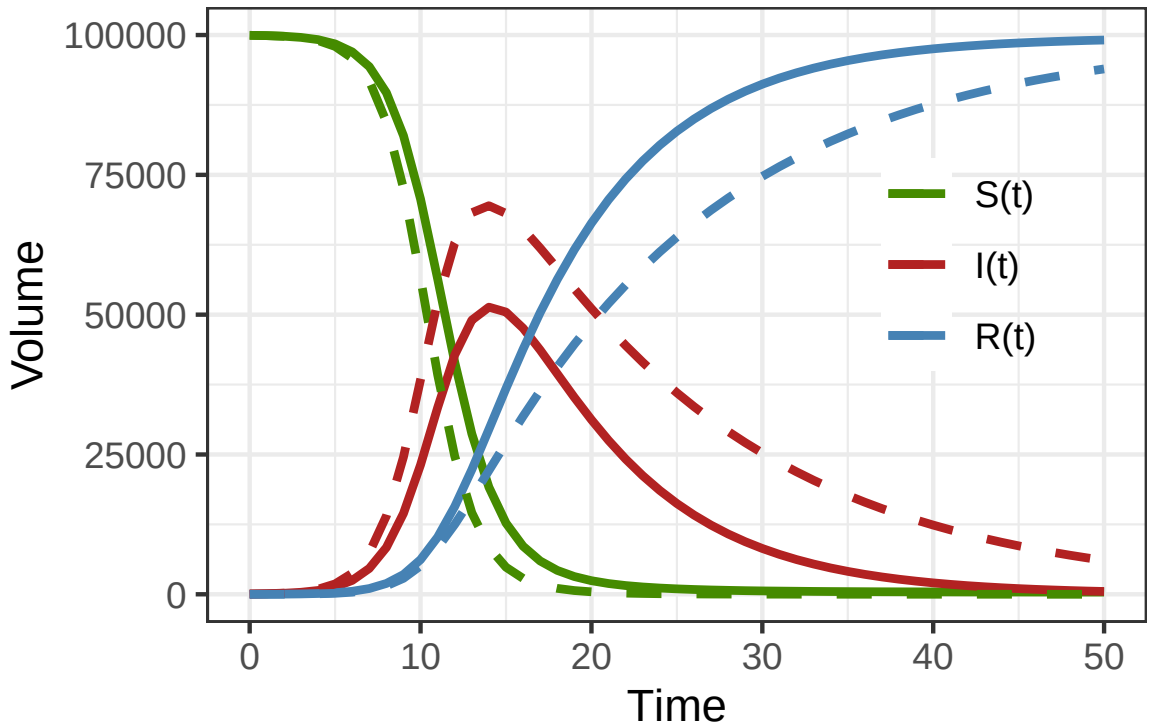
Simulate a SIR in R

with $\beta = 0.6$ instead of 0.8, we get



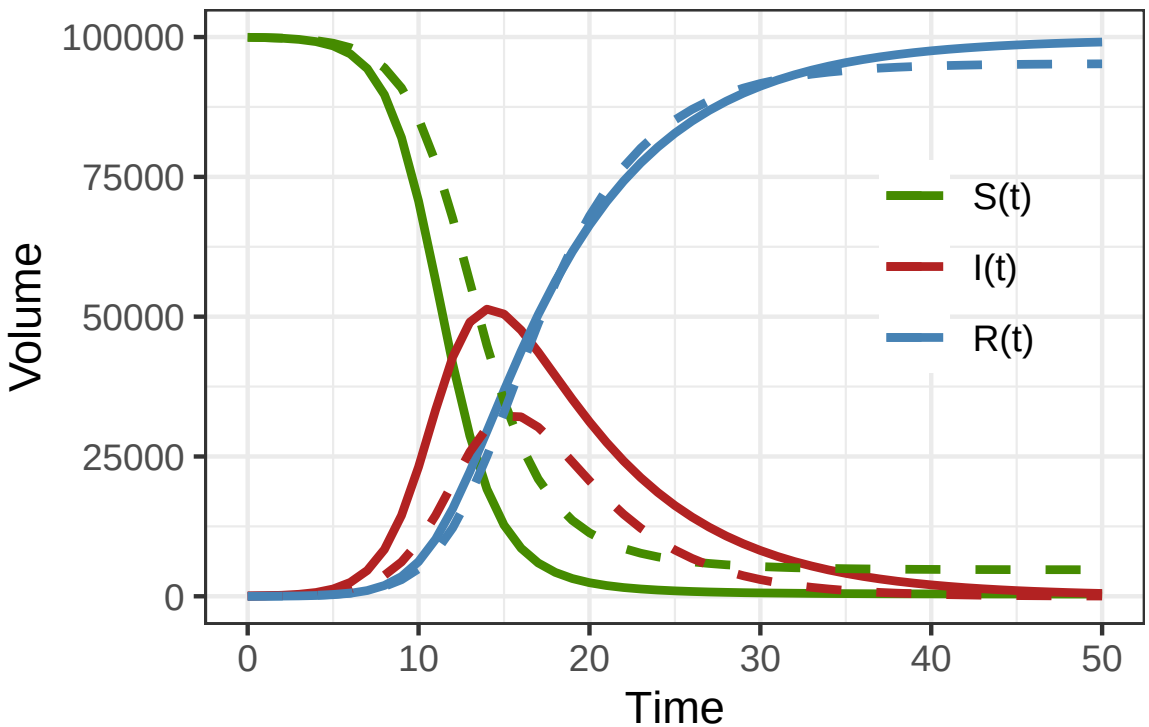
Simulate a SIR in R

with $\gamma = 1/14$ instead of $1/7$, we get



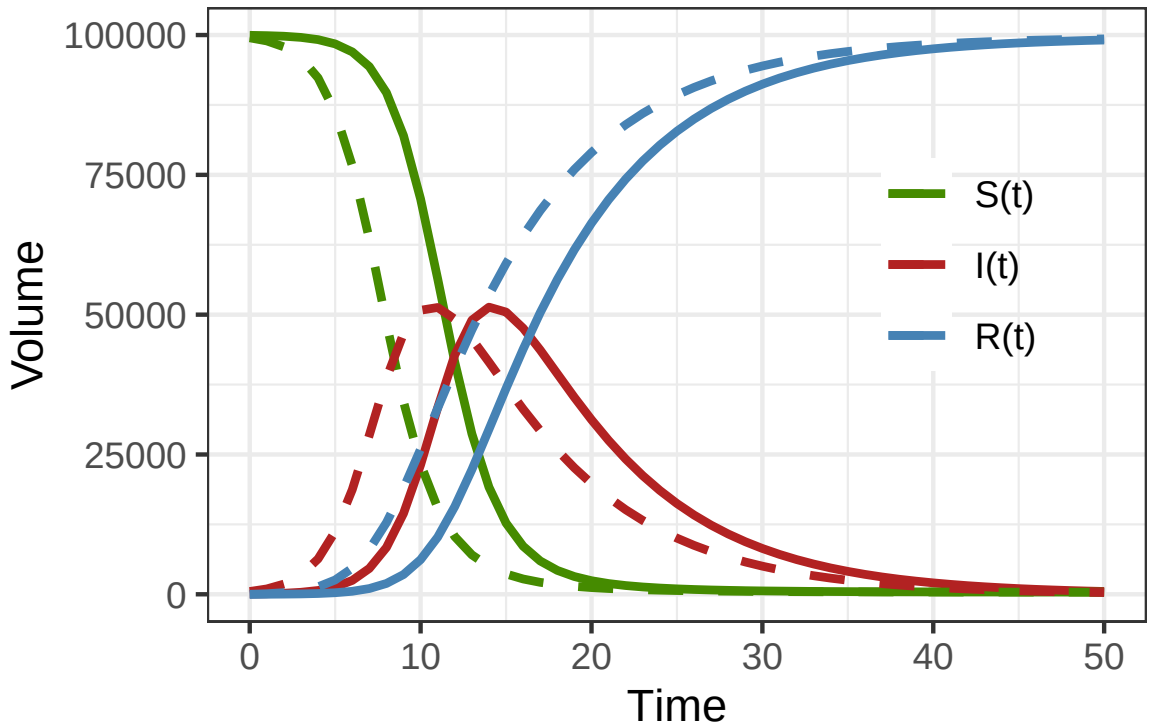
Simulate a SIR in R

with $\gamma = 1/4$ instead of $1/7$, we get



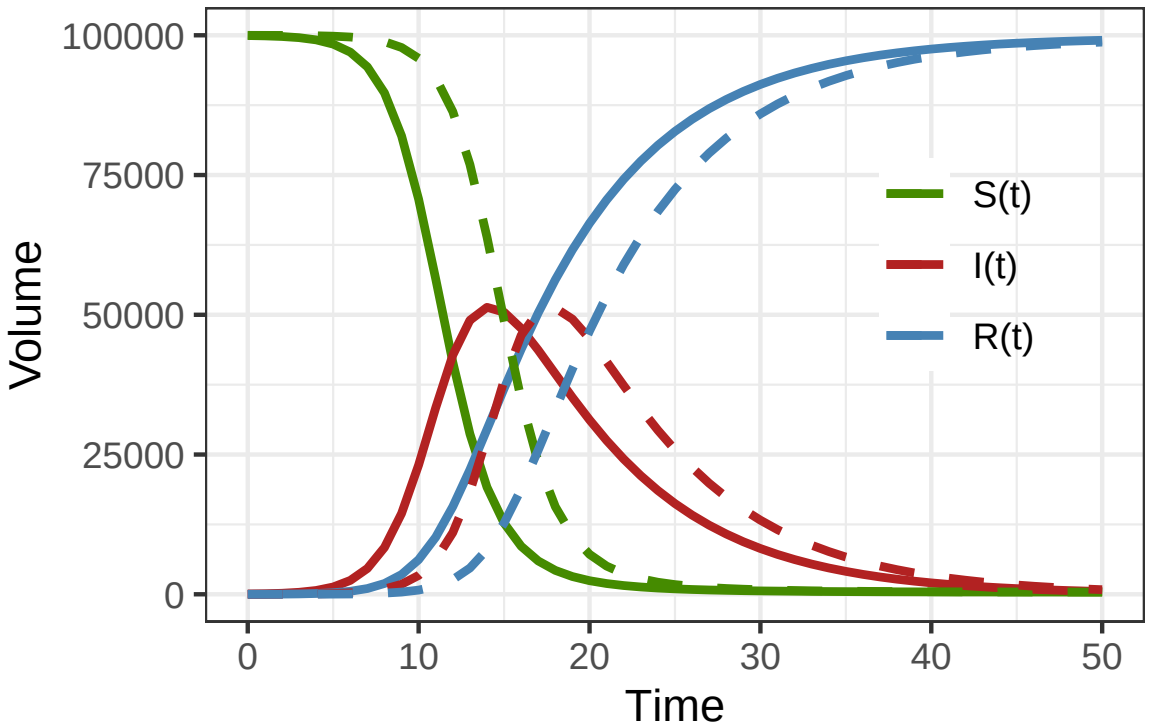
Simulate a SIR in R

with $I(0) = 500$ instead of 50, we get



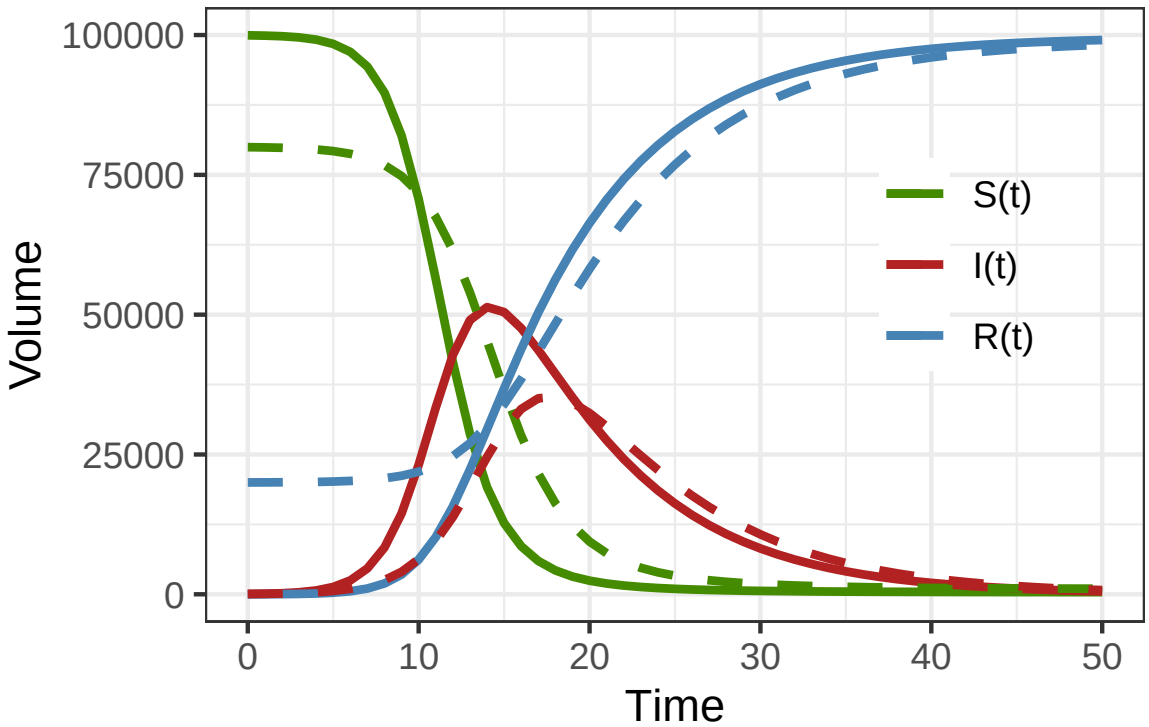
Simulate a SIR in R

with $I(0) = 5$ instead of 50, we get



Simulate a SIR in R

with $R(0) = 20,000$ instead of 0, we get



Applications

Compartmental models have many uses:

- get mechanistic insight about an epidemic (transmissibility levels, drivers of transmission)

$$\mathcal{R}_0 = \frac{\beta}{\gamma}$$

- formalize and put numerical values on general concepts (herd immunity threshold...)

$$V_c = \frac{1}{\mathcal{R}_0}$$

- produce (short-term) forecasts accounting for contagion and immunity

Applications

Compartmental models have many uses:

- get mechanistic insight about an epidemic (transmissibility levels, drivers of transmission)

$$\mathcal{R}_0 = \frac{\beta}{\gamma}$$

- formalize and put numerical values on general concepts (herd immunity threshold...)

$$V_c = \frac{1}{\mathcal{R}_0}$$

- produce (short-term) forecasts accounting for contagion and immunity

$\rightarrow$ all these uses are based on numerical values for β, γ and the initial conditions and their uncertainty

Applications

Enters Bayesian inference:

- infer parameter values by integrating data and domain knowledge
- more efficient for complex models (high dimensionality)
- rigorously quantify and propagate uncertainty in parameter estimates and forecast

Applications

Enters Bayesian inference:

- infer parameter values by integrating data and domain knowledge
- more efficient for complex models (high dimensionality)
- rigorously quantify and propagate uncertainty in parameter estimates and forecast

$\rightarrow$ Markov Chain Monte Carlo (MCMC) methods and Stan

Outline

Bayesian inference

$$\underbrace{p(\theta|y)}_{\text{posterior}} \propto \underbrace{p(y|\theta)}_{\text{likelihood}} \underbrace{p(\theta)}_{\text{prior}}$$

General principle

General principle of Bayesian inference:

- specify a complete Bayesian model
  - consider data $y = \{y_1, ..., y_n\}$ and parameter θ
  - specify an observation model, e.g.

$$p(y|\theta) = \prod_n \text{normal}(y_n|\theta, 1)$$

  - complete the model with a prior distribution, e.g.

$$p(\theta) = \text{normal}(0, 1)$$

- sample the posterior distribution of the parameter

Stan

Stan is a probabilistic programming framework for Bayesian inference

- it is designed to let the user focus on modeling while inference happens under the hood
- object-oriented language (based on `C++`) that supports many operations, probability densities and ODE solvers
- extremely efficient MCMC algorithm (Hamiltonian Monte Carlo)
- diagnostic tools to evaluate the inference
- interfaces in `R` (package `rstan`), `python`, `julia`...

Stan program structure

A Stan program is structured in blocks:

- `functions`
- `data`
- `transformed data`
- `parameters`
- `transformed parameters`
- `model`
- `generated quantities`

Stan example

- the `data` block defines data variables

```
data {
  int N;
  real y[N];
}
```

- the `parameters` block defines parameters

```
parameters {
  real theta;
}
```

- the `model` block defines the target log probability density function

```
model {
  theta ~ normal(0,1);
  y ~ normal(theta,1);
}
```

- save in `model_linear.stan`

Stan example

We then explore the target with Stan's MCMC sampler:

- load `rstan` package

```
## Setup ----
library(rstan)
options(mc.cores = parallel::detectCores())
```

- simulate $N = 50$ data points with $\theta = 0.7$

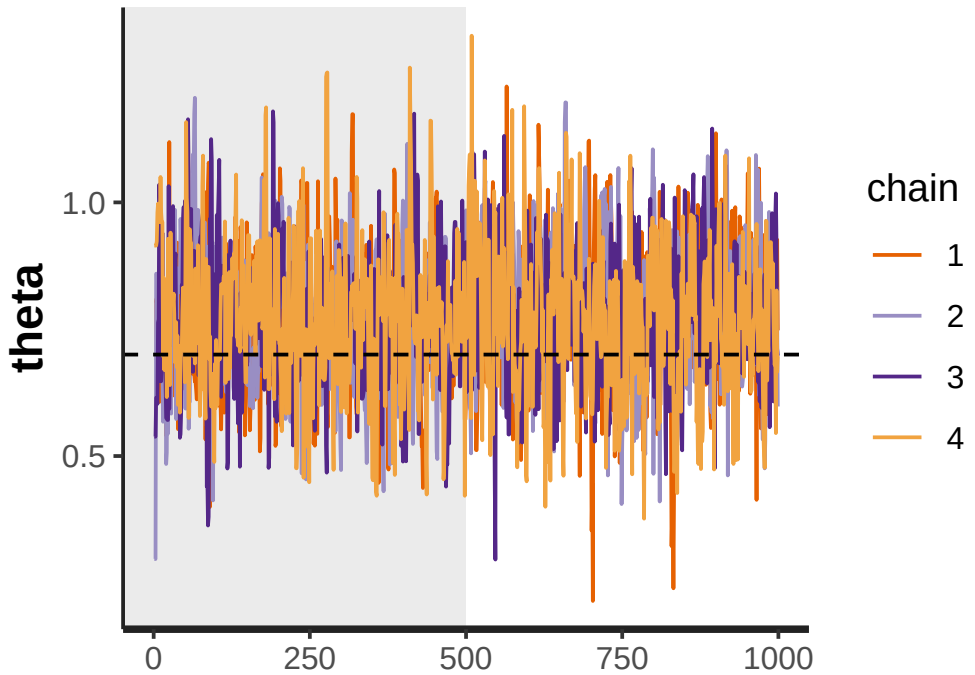
```
## Simulate data ----
N = 50
theta = 0.7
y = rnorm(N,theta,1)
input_data = list(N=N,y=y)
```

- run MCMC sampling

```
## Sample ----
fit = stan(file='model_linear.stan',
           data=input_data,
           chains=4,
           iter=1000)
```

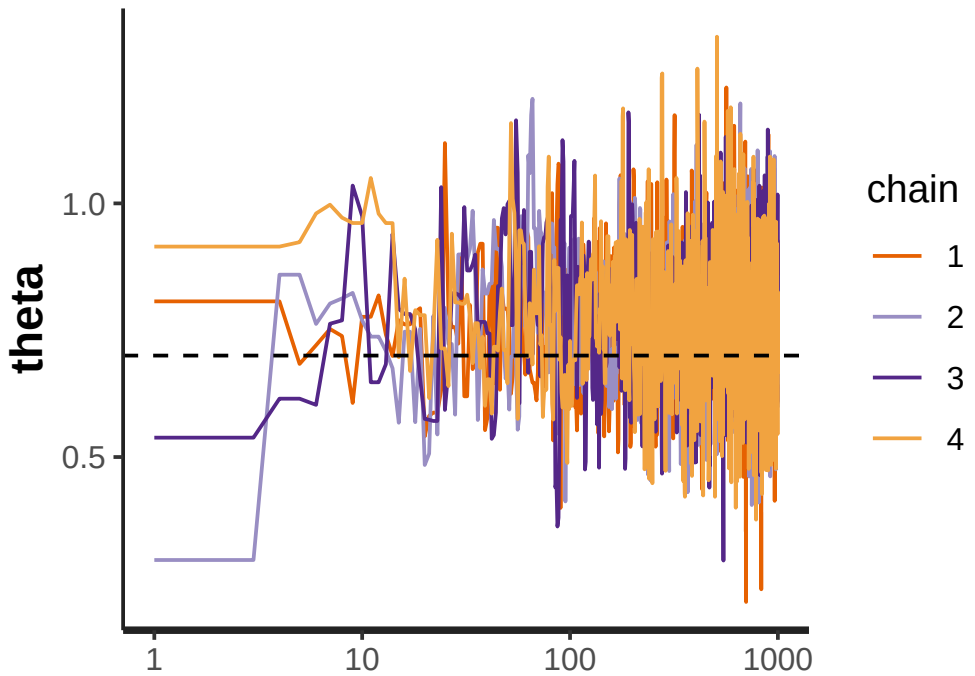
Stan example

We use multiple chains and inspect convergence after warm-up



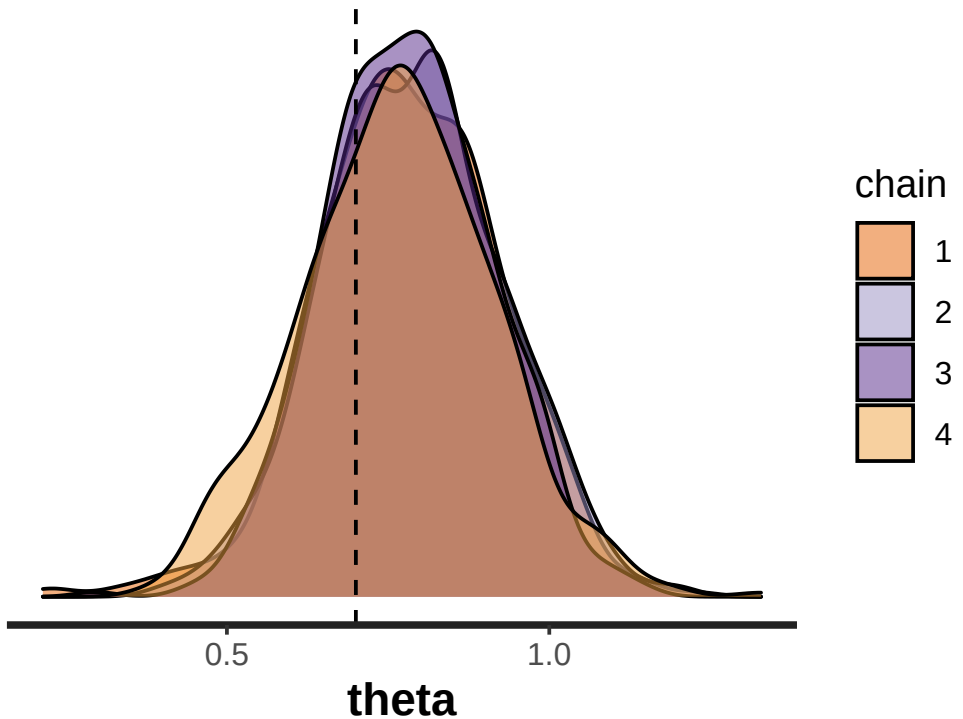
Stan example

We use multiple chains and inspect convergence after warm-up



Stan example

The post-warm-up samples of θ approximate its posterior distribution



Stan example

We run basic diagnostics tools: divergences, tree depth, energy

```
> check_hmc_diagnostics(fit)

Divergences:
0 of 2000 iterations ended with a divergence.

Tree depth:
0 of 2000 iterations saturated the maximum tree depth of 10.

Energy:
E-BFMI indicated no pathological behavior.
```

Stan example

Printing the object gives:

- diagnostics: effective sample size, Gelman-Rubin $\hat{R}$
- inference: full posterior distribution of θ

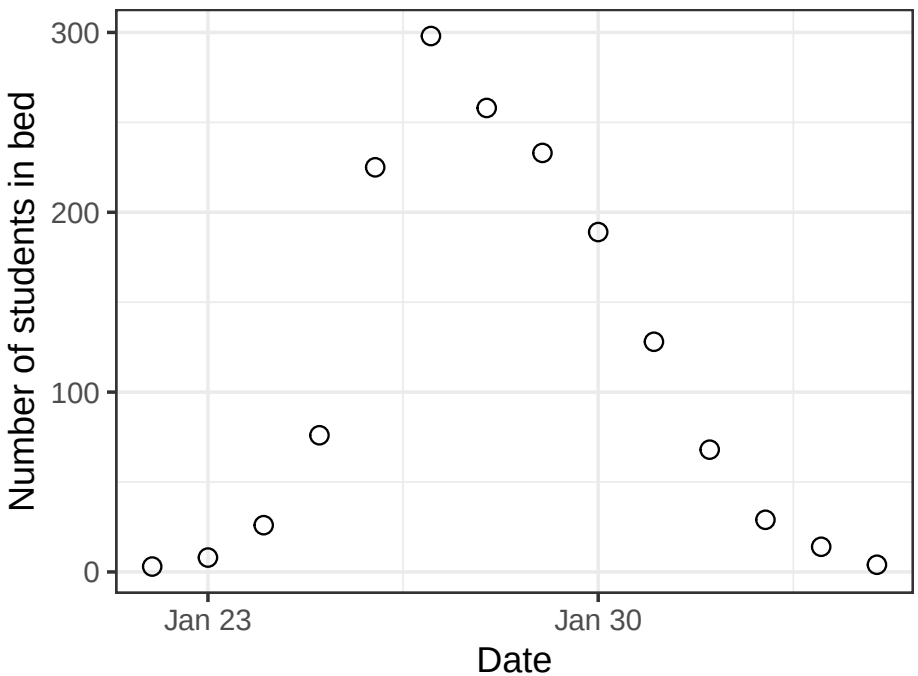
```
> print(fit)
Inference for Stan model: model_linear.
4 chains, each with iter=1000; warmup=500; thin=1;
post-warmup draws per chain=500, total post-warmup draws=2000.

        mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
theta   0.78    0.01 0.14   0.51   0.69   0.78   0.87   1.05   760    1
lp__  -18.30    0.03 0.73 -20.28 -18.44 -18.03 -17.86 -17.81   799    1

Samples were drawn using NUTS(diag_e) at Thu Nov 12 19:15:33 2020.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Outline

Example dataset

Outbreak of influenza A (H1N1) at a British boarding school in 1978 (available in `R` package `outbreaks`)

- 763 students, 512 had symptoms
- daily number of students in bed over 14 days (prevalence)



Specify the model

Points to consider:

- prevalence data: $\mathbb{I}_t$ with $t \in \{1, \ldots, 14\}$
- inputs that will remain fixed: $\{S_0 = 762, I_0 = 1, R_0 = 0\}$
- map data $\mathbb{I}_t$ to SIR model output $I(t)$ using an observation model with an appropriate probability distribution:

$$p(\mathbb{I}|\theta) = \prod_{t=1}^{14} \text{NegBinomial}(\mathbb{I}_t|I(t), \phi)$$

- parameters to estimate: $\theta = \{\beta, \gamma, \phi\}$
- prior distributions

$$p(\beta) = \text{Exponential}(1)$$

$$p(1/\gamma) = \text{Normal}(2, 0.5)$$

$$p(1/\phi) = \text{Exponential}(5)$$

Code the model

⚠ Note that further we are using ODE signature as presented in the paper. Since then a new interface has become available.

Code the model

We define the ODE system in the `function` block

```
functions {
  real[] sir(real t, real[] y, real[] theta, real[] x_r, int[] x_i) {

    real S = y[1];
    real I = y[2];
    real R = y[3];
    real N = x_i[1];

    real beta = theta[1];
    real gamma = theta[2];

    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};
  }
}
```

⚠ Be careful of the signature and formats!

Code the model

We declare the data variables in the `data` block

```
data {
  int<lower=1> T;
  real y0[3];
  real t0;
  real ts[T];
  int N;
  int cases[T];
}
```

Code the model

We declare the data variables in the `data` block

```
data {
  int<lower=1> T;
  real y0[3];
  real t0;
  real ts[T];
  int N;
  int cases[T];
}
```

and define additional data variables in `transformed data` block

```
transformed data {
  real x_r[0];
  int x_i[1];
  x_i[1]=N;
}
```

Code the model

Similarly, parameters are declared in the `parameters` block

```
parameters {
  real<lower=0> beta;
  real<lower=0> recovery_time;
  real<lower=0> phi_inv;
}
```

⚠ It sometimes makes more sense to transform some parameters (e.g., recovery rate γ and overdispersion ϕ) to improve interpretability

Code the model

In `transformed parameters`, we define additional parameters and solve the ODE system

```
transformed parameters{
  real y[T,3];
  real phi = 1. / phi_inv;
  real gamma = 1. / recovery_time;
  real theta[2];
  theta[1] = beta;
  theta[2] = gamma;

  y = integrate_ode_rk45(sir, y0, t0, ts, theta, x_r, x_i);
}
```

Code the model

```
y = integrate_ode_rk45(sir, y0, t0, ts, theta, x_r, x_i);
```

- Be careful about the formats and signatures
  - the ODE output `y` is an array of size `T`×3 (number of time steps and number of compartments)
  - `sir` is the name of the function defined in the `function` block
  - `y0` is an array of size 3 defined in the `data` block
  - `ts` is an array of size `T` defined in the `data` block
  - `theta` is an array of size 2 storing the parameters
  - `x_r` is defined as empty in `transformed data`, but can be used to store fixed real values
  - `x_i` is an array of size 1 storing the population size `N` (can also be used to store fixed integer values)

Code the model

```
y = integrate_ode_rk45(sir, y0, t0, ts, theta, x_r, x_i);
```

- Chose an ODE solver. In the previous interface:
  - `integrate_ode_rk45` uses the Runge-Kutta method (quicker but non-adapted to stiff systems)
  - `integrate_ode_bdf` uses the backward differentiation method (slower but adapted to stiff systems)
- In the new interface:
  - `ode_bdf`, `ode_adams`, `ode_rk45`,
  - `ode_bdf_tol`,`ode_adams_tol`, `ode_rk45_tol`

Code the model

In the `model` block, we write the priors and the observation model

```
model {
  // priors
  beta ~ exponential(1);
  recovery_time ~ normal(2,0.5);
  phi_inv ~ exponential(5);

  // observation model
  cases ~ neg_binomial_2(col(to_matrix(y),2), phi);
}
```

⚠ It's important that the chosen distributions correspond with the boundaries set in the `parameters` block (`<lower=0>`)

⚠ `col(to_to_matrix(y))` extracts the 2nd column of `y`

Code the model

Last, we add a `generated quantities` block that does not influence sampling and can be used for "post-processing":

- reproduction number $\mathcal{R}_0 = \beta/\gamma$
- model predictions of prevalence from the negative binomial

```
generated quantities {
  real R0 = beta/gamma;
  real pred_cases[T];
  pred_cases = neg_binomial_2_rng(col(to_matrix(y),2), phi);
}
```

Code the model

In summary:

- `functions`: define the ODE system (⚠ signature and formats)
- `data`: declare data variables that will be provided
- `tranformed data`: additional quantities that can be computed internally or from `data` variables
- `parameters`: declare parameters (⚠ boundaries)
- `transformed parameters`: quantities that can be computed internally or from `data` or `parameters` variables, including the ODE output (⚠ signature and format)
- `model`: priors and observation model
- `generated quantities`: additional quantities that can be computed without influencing the sampling

Inference

As before, we conduct the inference from `R` with the package `rstan`:

```r
## Format input ----
# prevalence data
cases = influenza_england_1978_school$in_bed
N = 763
n_days = 14
t0 = 0
t = 1:n_days

# initial conditions
i0 = 1
s0 = N - i0
r0 = 0
y0 = c(s0, i0, r0)

# put into list
input_data = list(T = n_days, y0 = y0, t0 = t0, ts = t, N = N, cases = cases)
```

⚠ data is put in a list with names matching the `data` block in Stan

Inference

Hit the inference button!

```
## Sample ----
fit = stan(file='sir_negbin.stan',
           data=input_data,
           chains=4,
           iter=1000)
```

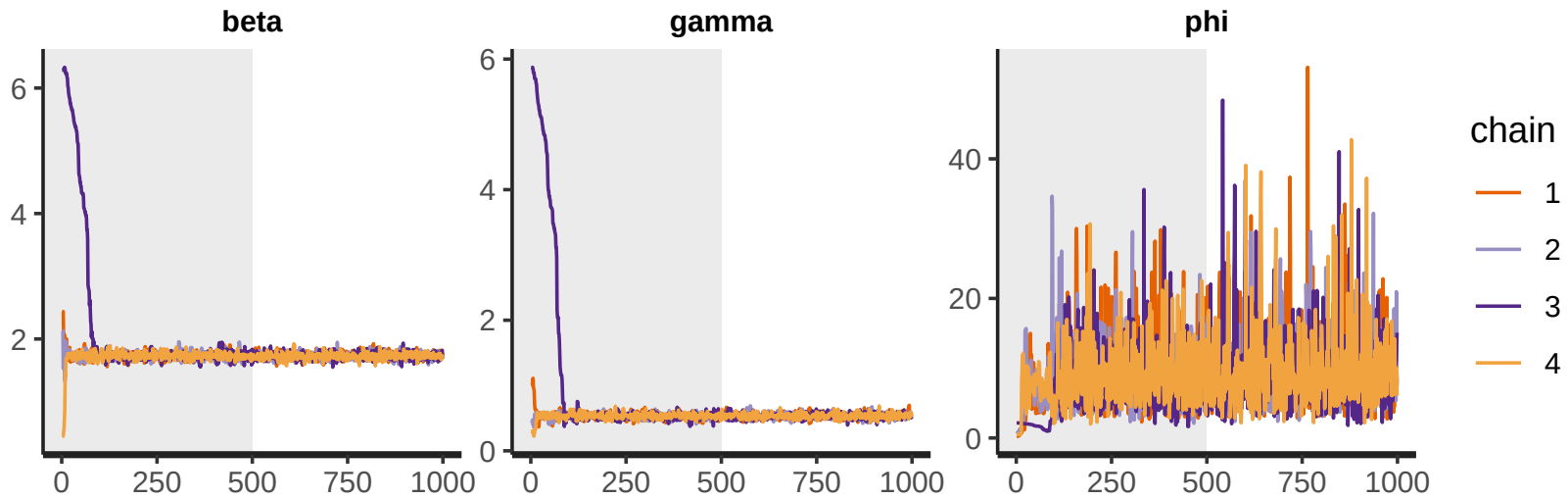
Diagnostics

Use the basic diagnostics tools:

```
> check_hmc_diagnostics(fit)

Divergences:
0 of 2000 iterations ended with a divergence.

Tree depth:
0 of 2000 iterations saturated the maximum tree depth of 10.

Energy:
E-BFMI indicated no pathological behavior.
```

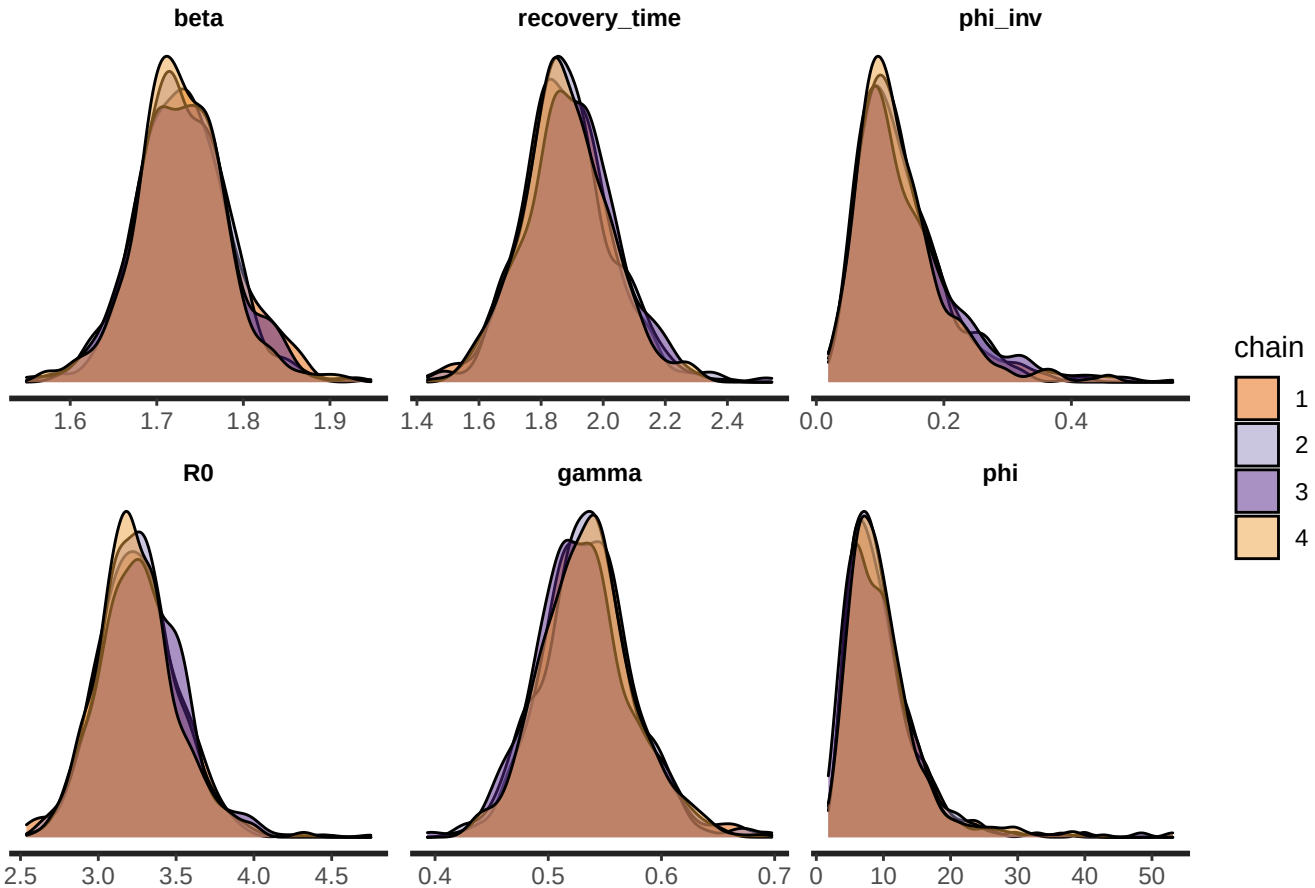
Diagnostics

Examine trace plots:



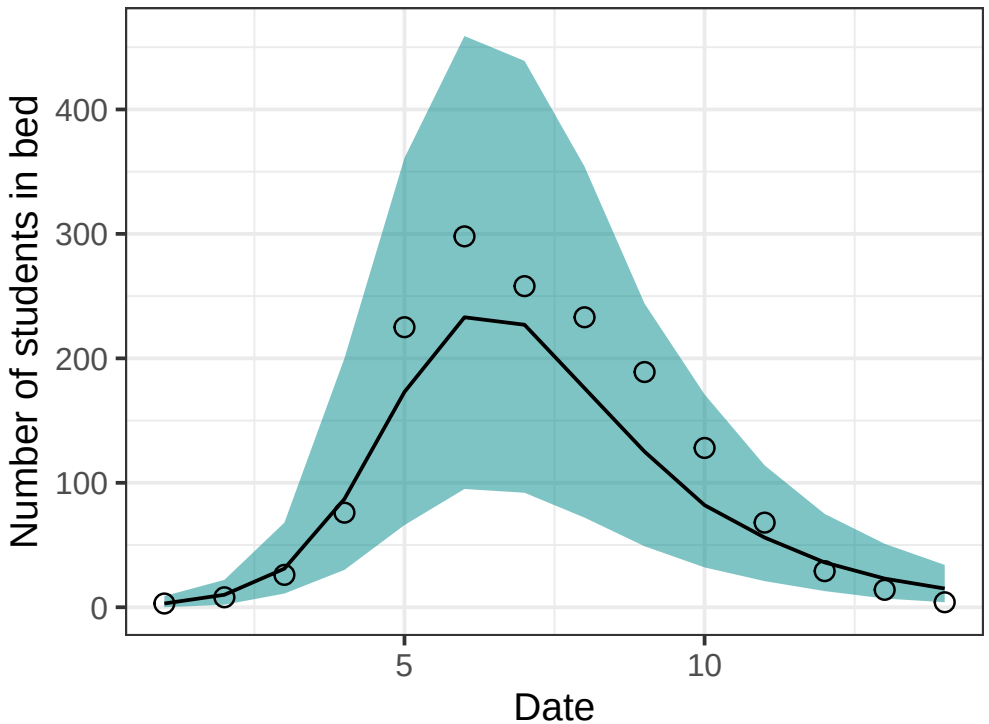
Diagnostics

Examine chain mixing:



Diagnostics

Posterior predictive checking:



Results

Print the results:

```
> print(fit,pars=c("beta","gamma","phi","R0","recovery_time"))
Inference for Stan model: sir_negbin.
4 chains, each with iter=1000; warmup=500; thin=1;
post-warmup draws per chain=500, total post-warmup draws=2000.

              mean se_mean   sd 2.5%  25%  50%   75% 97.5% n_eff Rhat
beta          1.73    0.00 0.06 1.61 1.70 1.73  1.76  1.85  1049    1
gamma         0.53    0.00 0.04 0.44 0.50 0.53  0.56  0.63  1382    1
phi           9.61    0.22 6.13 2.94 5.72 8.31 11.80 23.40   743    1
R0            3.27    0.01 0.29 2.79 3.09 3.25  3.42  3.96  1403    1
recovery_time 1.89    0.00 0.16 1.59 1.79 1.88  1.98  2.25  1410    1

Samples were drawn using NUTS(diag_e) at Thu Nov 12 19:10:02 2020.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Results

- we estimate $\mathcal{R}_0$ to 3.3 (95% credible interval: 2.8 to 4.0)
- this corresponds to the direct estimation from the final size of the epidemic $q = 512/763 = 0.67$

$$\mathcal{R}_0 = 1/(1 - q) = 3.03$$

- based on many assumptions:
  - common to all SIRs (homogeneous mixing, no incubation...)
  - prior distributions (especially on the recovery period)
  - complete ascertainment, no asymptomatics
  - no initial immunity

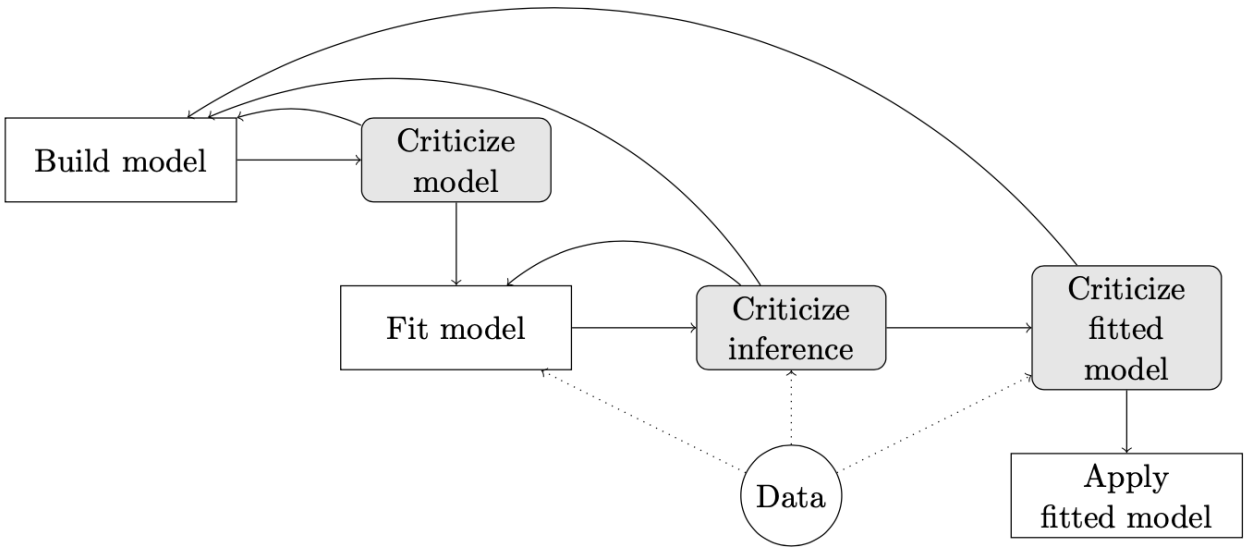
Outline

Box's loop

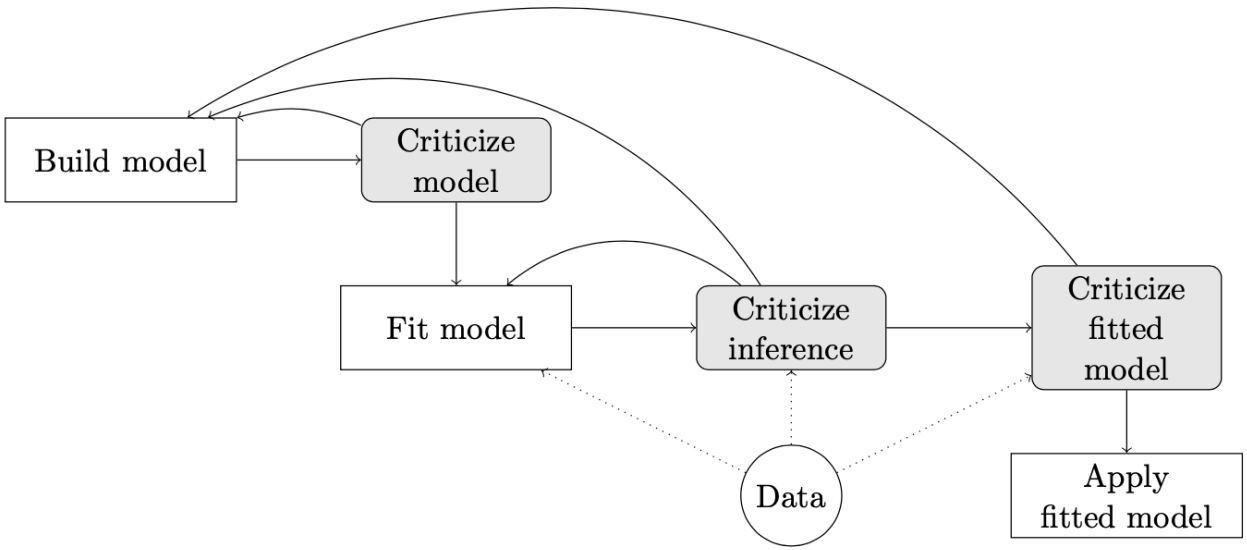
In practice, the situation is often less clear than in the boarding school example:

- incomplete data, insufficient domain knowledge
- uncertainty on necessary model features

Box's loop


Build model
Criticize model
Fit model
Criticize inference
Criticize fitted model
Data
Apply fitted model

Box's loop


Build model
Criticize model
Fit model
Criticize inference
Criticize fitted model
Data
Apply fitted model

Model development

Model development is an iterative procedure:

- troubleshoot the model before fitting it,
- criticize the inference after attempting a fit,
- criticize the fitted model.

Outline

Model development

Fake data can be used to probe the model and better understand its behaviour:

- prior predictive checks
- simulation study

Prior predictive checks

Prior predictive checking consists in simulating data from the priors:

- visualize priors (especially after transformation)
- this shows the range of data compatible with the model
- it helps understand the adequacy of the chosen priors, as it is often easier to elicit expert knowledge on measureable quantities of interest rather than abstract parameter values

Prior predictive checks

Prior predictive checking consists in simulating data from the priors:

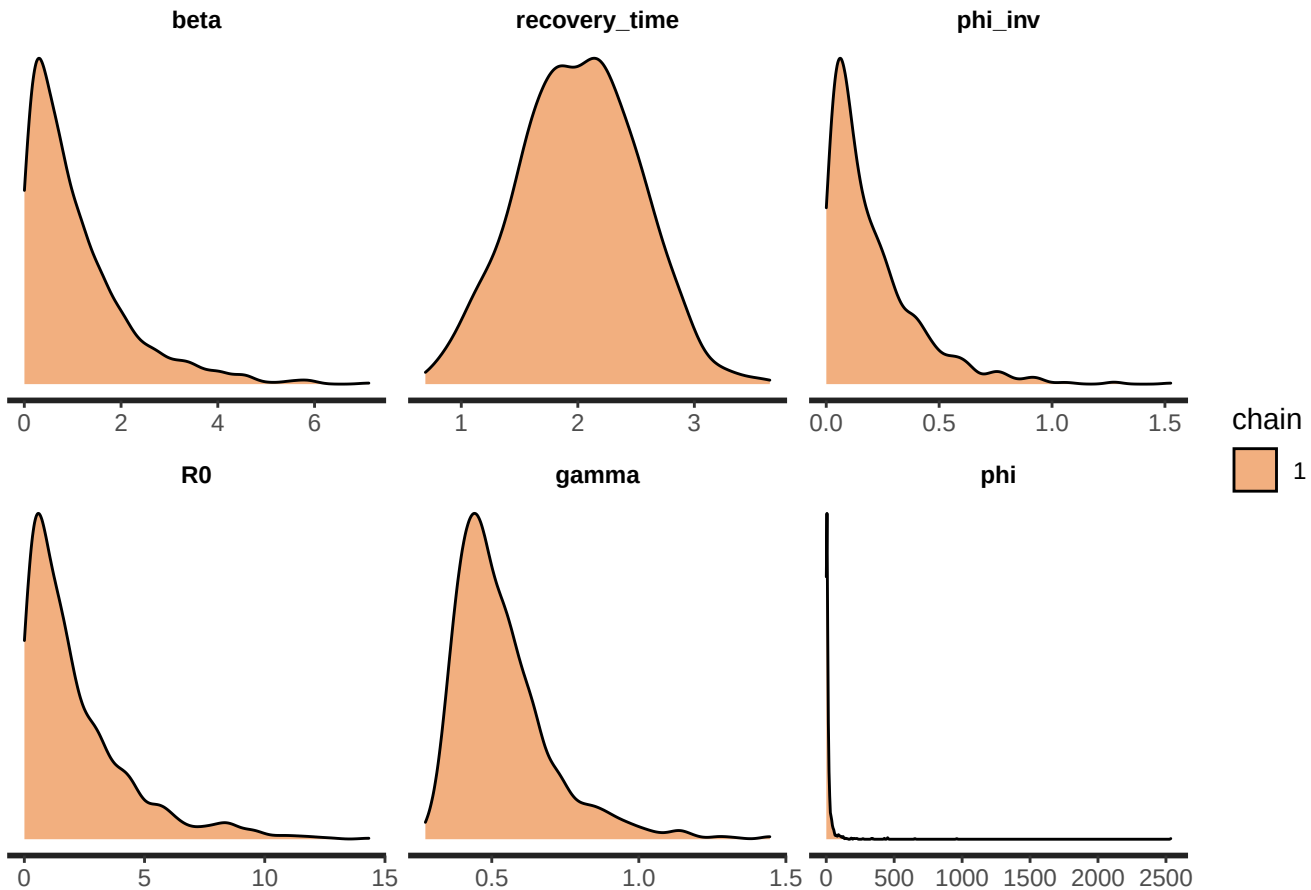
- visualize priors (especially after transformation)
- this shows the range of data compatible with the model
- it helps understand the adequacy of the chosen priors, as it is often easier to elicit expert knowledge on measureable quantities of interest rather than abstract parameter values

- remove (or switch off) the likelihood from the `model` block

```
// observation model
if(switch_likelihood==1) cases ~ neg_binomial_2(col(to_matrix(y),2), phi);
```

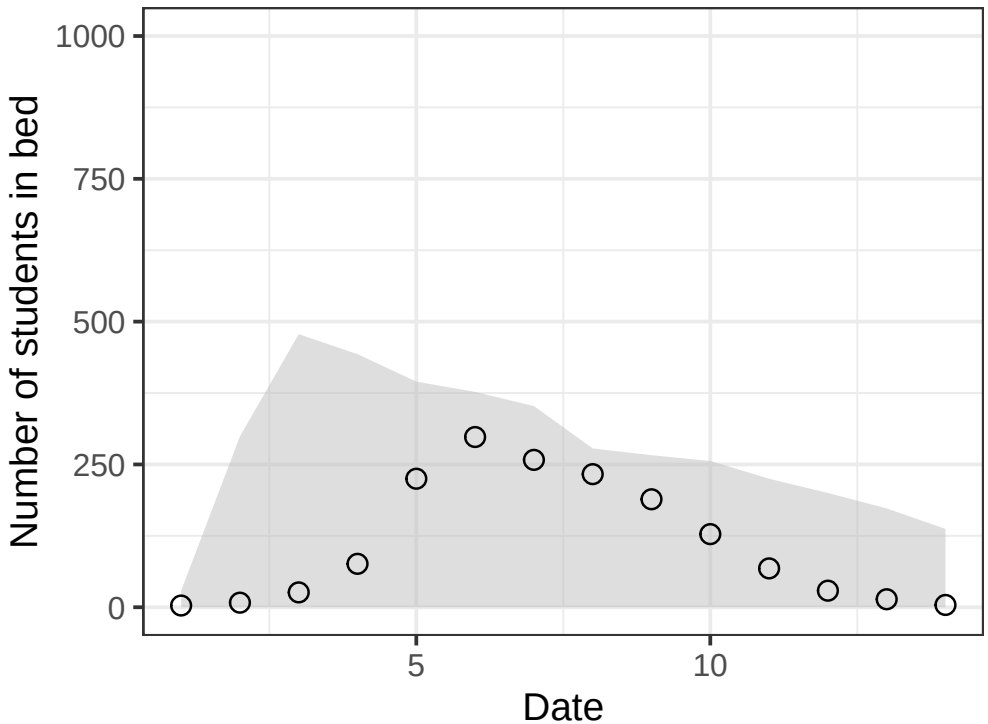
Prior predictive checks

Simulating priors in the boarding school example:



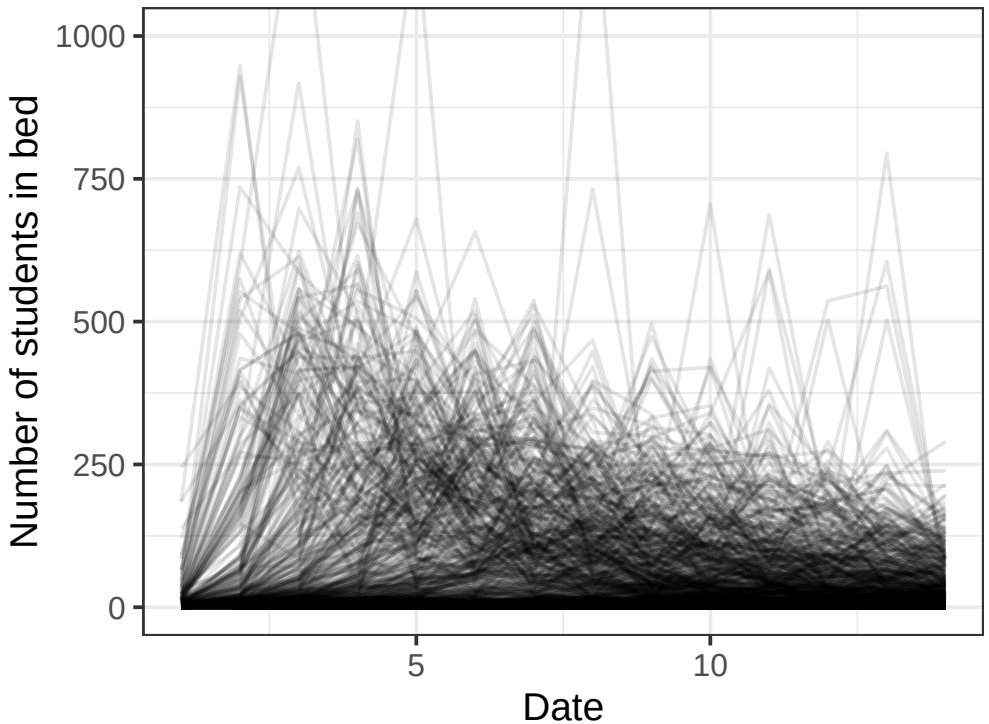
Prior predictive checks

Prior predictive check: simulating potential epidemic trajectories



Prior predictive checks

Prior predictive check: simulating potential epidemic trajectories



Prior predictive checks

Prior predictive checks bring insight about non-obvious features:

- even if the priors seem weakly informative, there is actually not a lot of leeway
- highly constrained model:
  - if β is high, the epidemic will stop rapidly by lack of susceptibles
  - if β is small, the epidemic will be small
- the negative binomial distribution might lead to problems in extreme situations, e.g. more cases ($>$1000) than the overall number of students

Simulation-based checks

A simulation study consists out of two steps:

- simulate data with specified parameter values
- measure the capacity of the model to recover the chosen parameter values

Simulation-based checks

A simulation study consists out of two steps:

- simulate data with specified parameter values
- measure the capacity of the model to recover the chosen parameter values

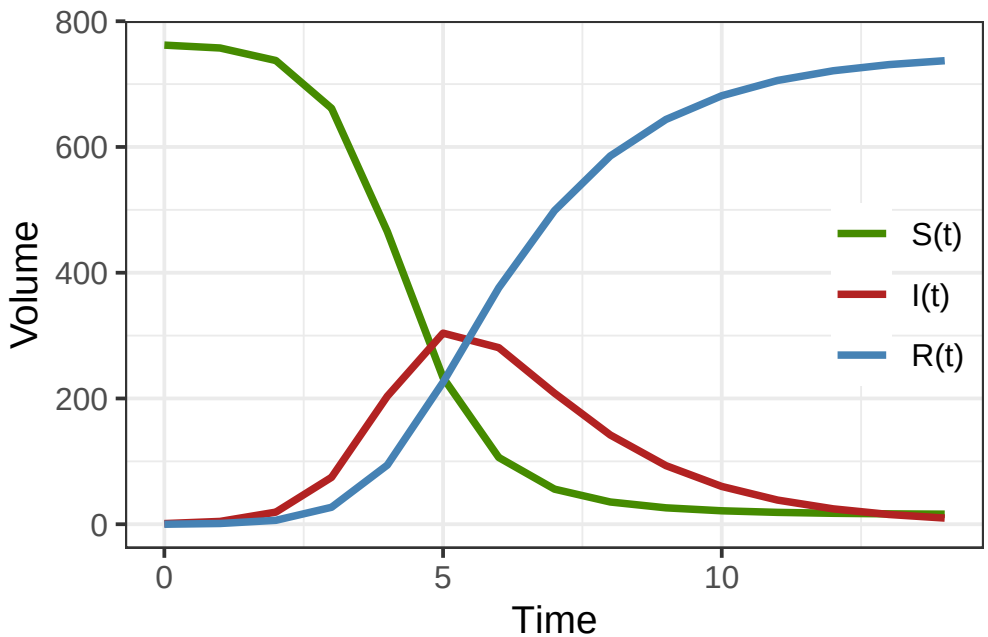
Many advantages:

- check for bugs and coding mistakes
- check for identifiability issues
- compare different versions of a model
- understand in what situations a model works or not

Simulation-based checks

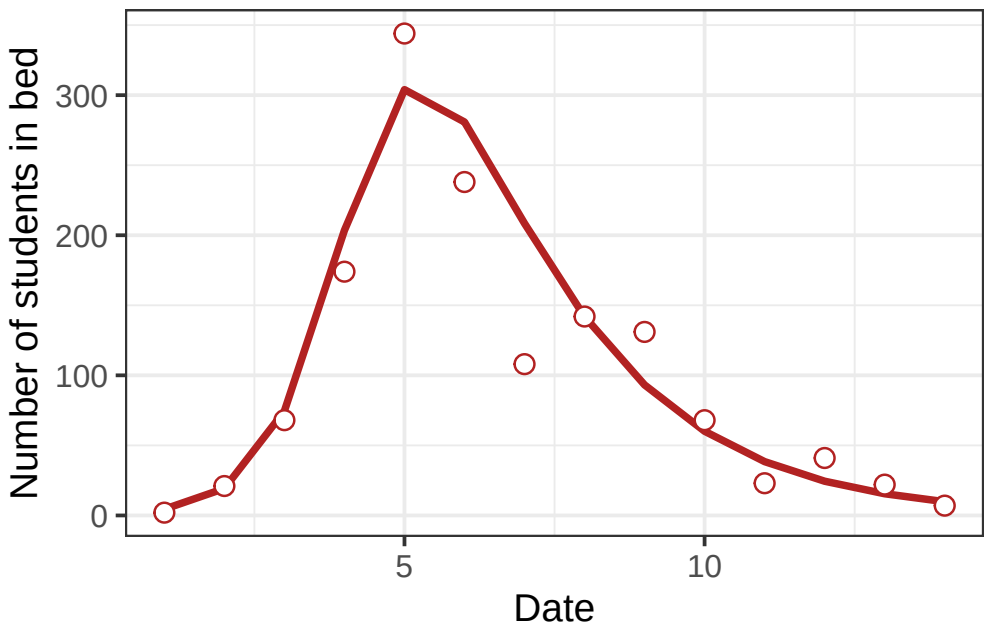
Let's go back to the simple SIR example from the beginning:

- set $\beta = 2$ and $\gamma = 0.5$ (so that $\mathcal{R}_0 = 4$)
- simulate in a susceptible population of size $N = 763$ with $I_0 = 1$



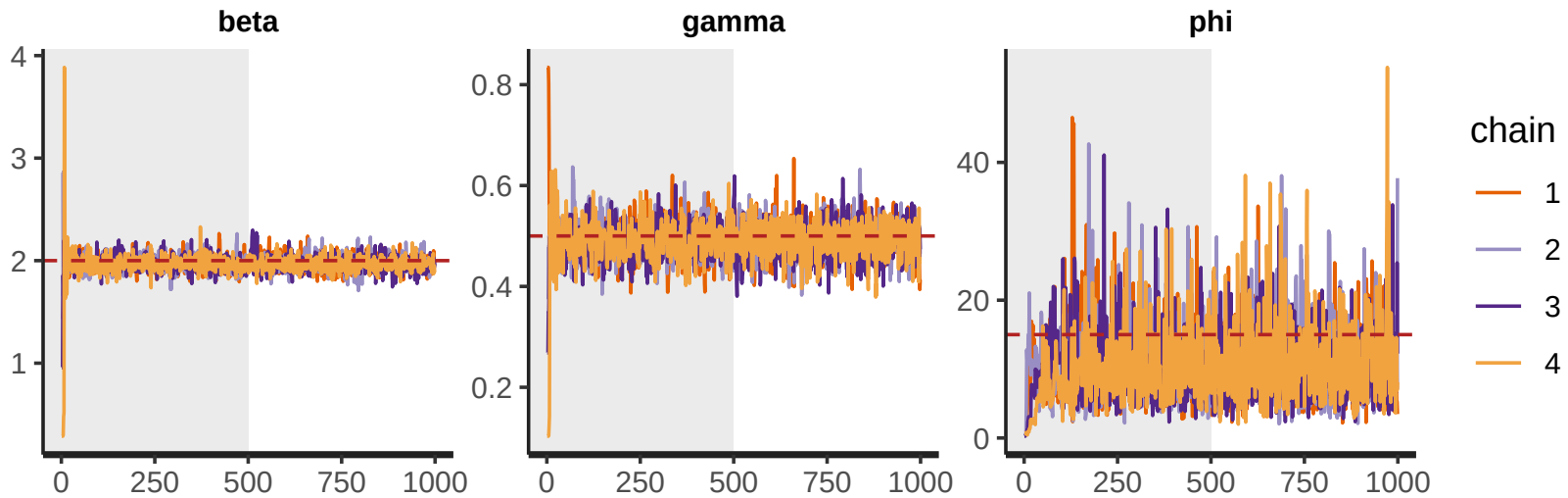
Simulation-based checks

Add noise with a negative binomial distribution with $\phi = 15$



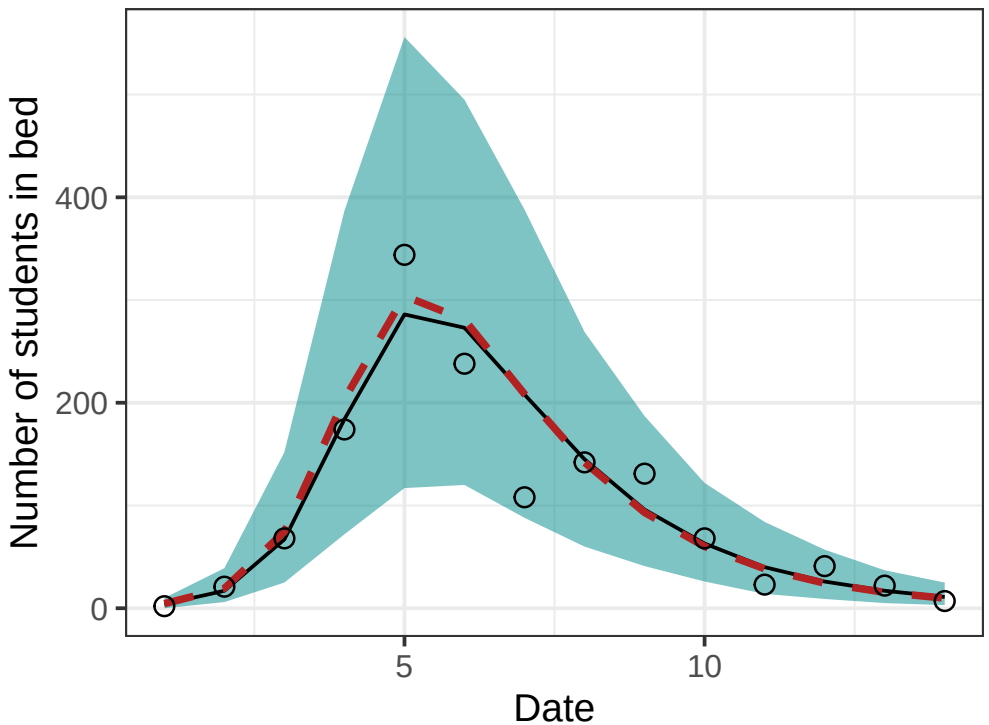
Simulation-based checks

Fit the same model as for the boarding school example



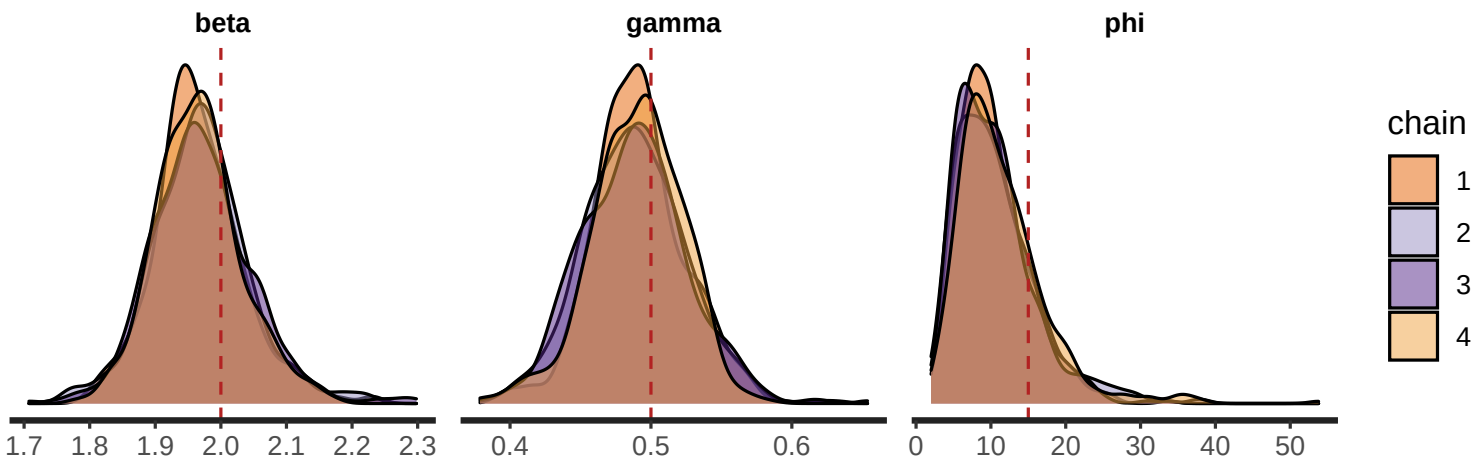
Simulation-based checks

Posterior predictive checking



Simulation-based checks

Compare the posterior distributions of the parameters with the "truth"



- no identifiability issue
- β and γ are well estimated, but ϕ is not
- try with other values to understand when does the model break

Outline

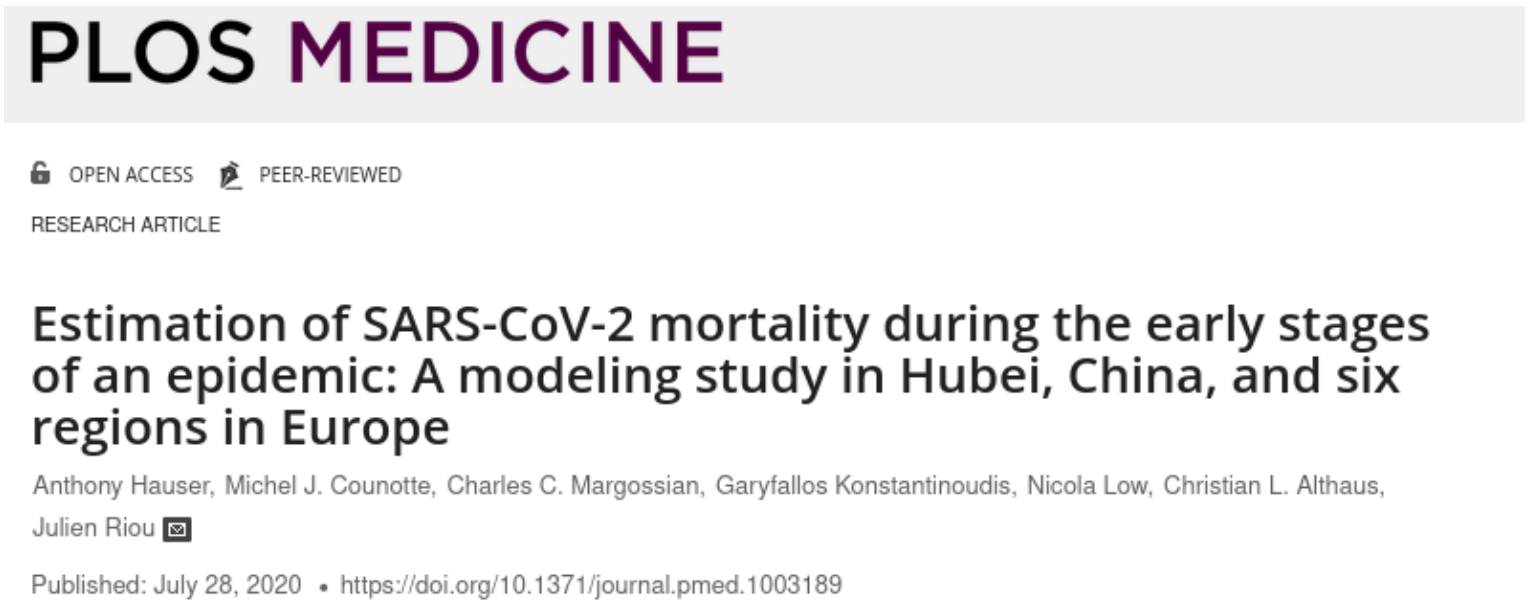
Limitations to the SIR model

In practice, the boarding school example quickly reaches its limits:

- epidemics are not often observed in such a controlled environment (under-ascertainment)
- epidemics are not always left uncontrolled
- data generally consist of daily counts of new cases (incidence) rather than counts of currently sick people (prevalence)
- most infectious diseases have an incubation period (SEIR instead of SIR)
- transmission is generally not homogeneous in the full population (stratification by age, sex...)
- ...

Example

Example with SARS-CoV-2 from Hauser et al. (2020):



PLOS MEDICINE

OPEN ACCESS PEER-REVIEWED

RESEARCH ARTICLE

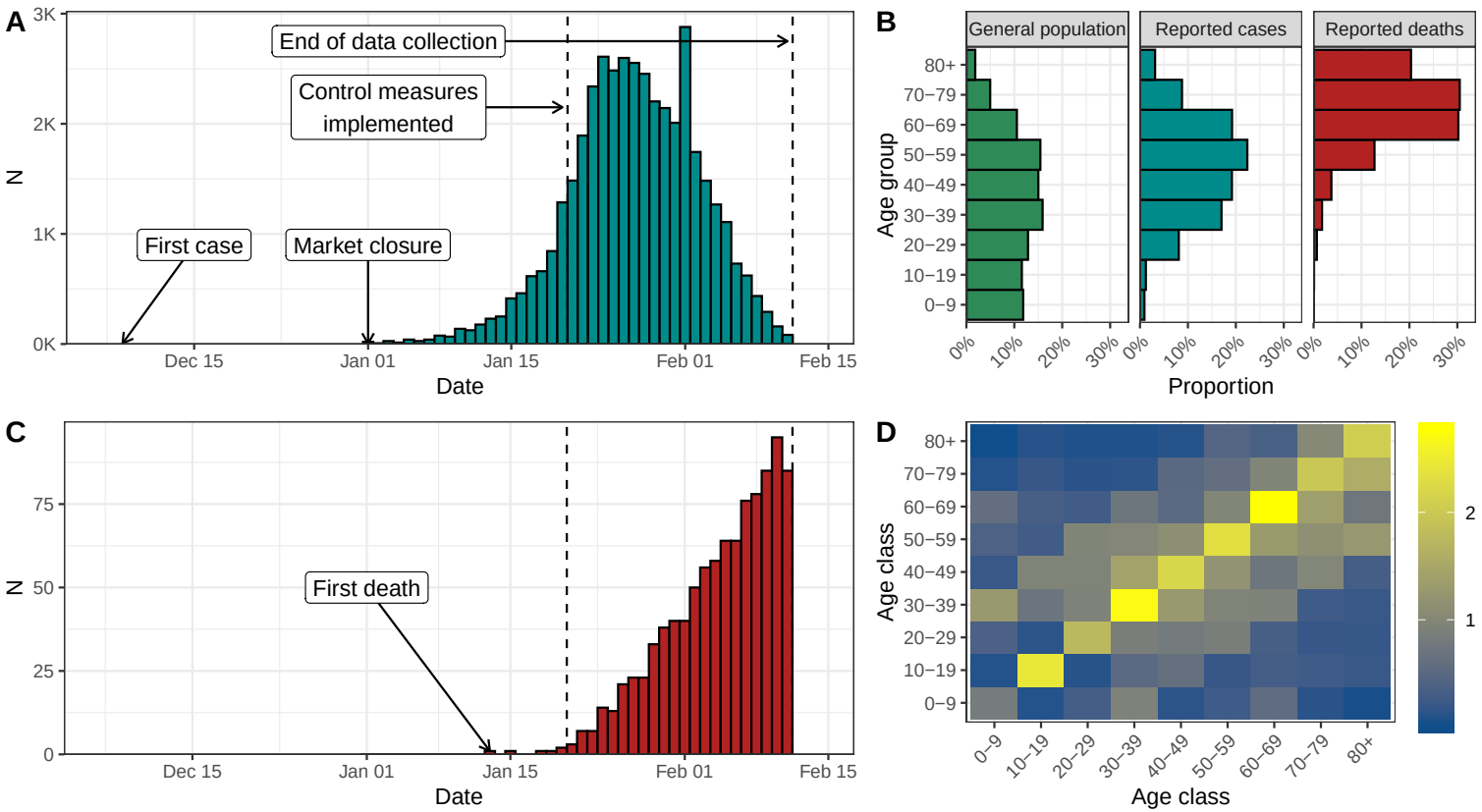
Estimation of SARS-CoV-2 mortality during the early stages of an epidemic: A modeling study in Hubei, China, and six regions in Europe

Anthony Hauser, Michel J. Counotte, Charles C. Margossian, Garyfallos Konstantinoudis, Nicola Low, Christian L. Althaus, Julien Riou

Published: July 28, 2020 • https://doi.org/10.1371/journal.pmed.1003189

Example

In Hubei province (China):



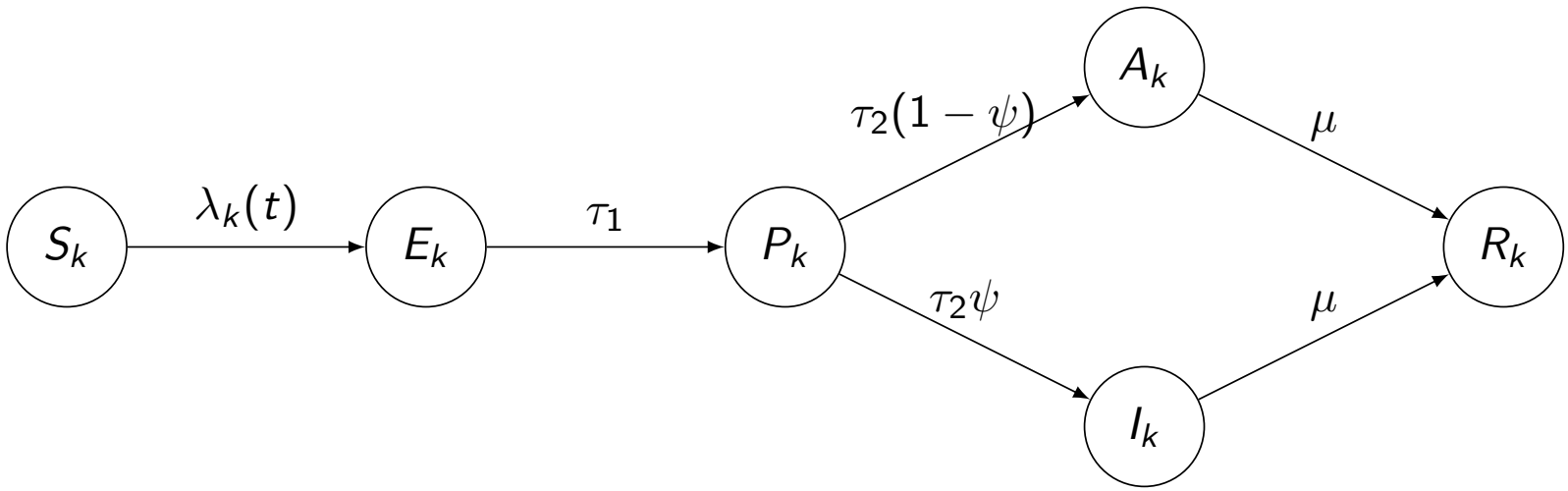
Example

Specific features and natural history of SARS-CoV-2 infection:

- Incubation period of 5 days (SEIR)
- Pre-symptomatic transmission accounting for 44-48% (SEPIR)
- Symptoms in 81% (95%CrI 71%–89%) of cases (SEPIAR)
- Respiratory virus transmitted through contacts (age stratification)
- Effect of control measures (time-dependent force of infection)
- Mortality is delayed by 20.2 ± 11.6 days (fit to mortality data)

Example

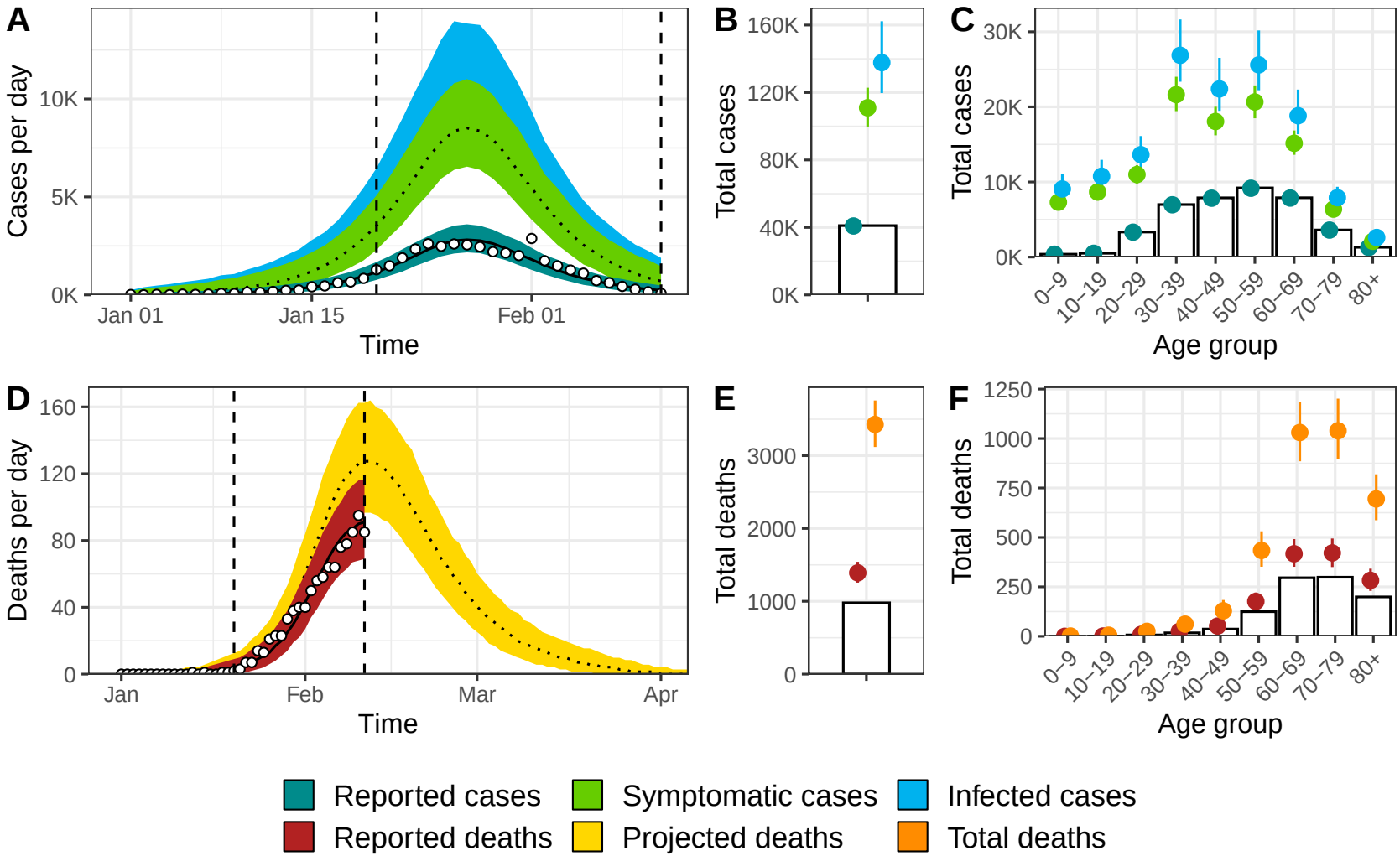
Final model:



S_k	Susceptible (for age group k)	$\lambda_k(t)$	Force of infection (time-dependent)
E_k	Exposed	$1/\tau_1 + 1/\tau_2$	Incubation period (split in two)
P_k	Presymptomatic	ψ	Proportion of symptomatic
A_k	Infected asymptomatic	$1/\tau_2$	Presymptomatic infectious period
I_k	Infected symptomatic	$1/\mu$	Symptomatic infectious period
R_k	Removed		

Example

Posterior predictive check (Hubei):



Outline

Conclusions

General comments:

- develop models that correspond to the data-generating mechanisms
- use Bayesian inference to propagate uncertainty from the data (and priors) into the results (and forecasts)
- carefully examine the modelling process (Bayesian workflow)
- be transparent about assumptions (open code)

Try by yourself!

- `https://github.com/jriou/bayesian_workflow_sir/tree/advanced_stat_physicists_2021`
- julien.riou@ispm.unibe.ch

Acknowledgements & ressources

- Michael Betancourt, *Introduction to Stan*
  `https://betanalpha.github.io/assets/case_studies/stan_intro.html`
- Andrew Gelman et al., *Bayesian workflow*
  `https://arxiv.org/abs/2011.01808`
- Chi Feng, *MCMC interactive gallery*
  `https://chi-feng.github.io/mcmc-demo/app.html`
- Daniel Lee, *ODEs in Stan*
  `https://youtu.be/hJ34_xJhYeY`
- Ben Bales, Sebastian Weber , *Upgrading to the new ODE interface*
  `https://mc-stan.org/users/documentation/case-studies/convert_odes.html`