

McMasterPandemic: getting started

Ben Bolker and David Earn
earn@math.mcmaster.ca

November 23, 2020 @ 12:46

Abstract

McMasterPandemic is an R package that provides tools for simulating and forecasting infectious disease outbreaks, using compartmental epidemic models. The primary mechanistic framework is a susceptible-exposed-infectious-removed (SEIR) model, with additional compartments for individuals in acute and intensive care units in hospitals.

Contents

1	Installation	1
2	Data requirements	2
3	Running a simulation	4
3.1	Stochasticity	5
3.2	Time-dependent transmission rate	8
4	Changing parameters	8
5	Calibration	9

1 Installation

Use `remotes::install_github("bbolker/McMasterPandemic")` to install the latest version of the package.

```
library(McMasterPandemic)

#> Warning: replacing previous import 'vctrs::data_frame' by 'tibble::data_frame'
when loading 'dplyr'
```

In this vignette we'll also use some other packages:

```
library(ggplot2); theme_set(theme_bw())  
library(cowplot)
```

2 Data requirements

Parameters To run simulations, a few parameter values must be specified. Set these by editing the example params file, which is converted to a `params_pansim` object by `read_params()`. In the example, the time unit is assumed to be days.

The term “in acute care” means “in hospital but not in the intensive care unit (ICU)”.

If `p` is a parameter set (e.g. the result of `read_params`), then `print(p, describe=TRUE)` will return a data frame with a column giving the meaning of each parameter.

```
params1 <- read_params("ICU1.csv")
```

(by default `read_params` looks first in the working directory for CSV files, then in the `params` directory installed with the package (`system.file("params", package="McMasterPandemic")`)).

symbol	value	meaning
beta0	1	Baseline (non-intervention) transmission across categories
Ca	0.667	relative asymptomatic transmission (or contact)
Cp	1	relative presymptomatic transmission (or contact)
Cm	1	relative mildly symptomatic transmission (or contact)
Cs	1	relative severely symptomatic transmission (or contact)
alpha	0.333	Fraction of cases asymptomatic
sigma	0.192	1/time in exposed class
gamma_a	0.143	1/time for asymptomatic recovery
gamma_m	0.143	1/time for mildly symptomatic recovery
gamma_s	0.175	1/time for severely symptomatic transition to hospital/death
gamma_p	2	1/time in pre-symptomatic class
rho	0.1	1/time in hospital (acute care)
delta	0	Fraction of acute-care cases that are fatal
mu	0.956	Fraction of symptomatic cases that are mild
N	1e+06	Population size
E0	5	Initial number exposed
nonhosp_mort	0	probability of mortality without hospitalization
iso_m	0	Relative self-isolation/distancing of mild cases
iso_s	0	Relative self-isolation/distancing of severe cases
phi1	0.76	Fraction of hospital cases to acute care
phi2	0.5	Fraction of ICU cases dying
psi1	0.05	Rate of ICU back to acute care
psi2	0.125	Rate of ICU to death
psi3	0.2	Rate of post-ICU to discharge
c_prop	0.1	fraction of incidence reported as positive tests
c_delay_mean	11	average delay between incidence and test report
c_delay_cv	0.25	coefficient of variation of testing delay
proc_disp	0	dispersion parameter for process error (0=demog stoch only)
zeta	0	phenomenological heterogeneity parameter

The `summary` method for `params_pansim` objects returns the initial exponential growth rate (r_0), the doubling time ($\log 2/r_0$), the mean generation interval (\bar{G}), and the basic reproduction number

$$\mathcal{R}_0 = \beta_0 \left\{ \alpha \frac{C_a}{\gamma_a} + (1 - \alpha) \left[\frac{C_p}{\gamma_p} + \mu(1 - \text{iso}_m) \frac{C_m}{\gamma_m} + (1 - \mu)(1 - \text{iso}_s) \frac{C_s}{\gamma_s} \right] \right\} .$$

```
knitr::kable(round(t(summary(params1)), 2))
```

r0	R0	Gbar	CFR_gen	dbl_time
0.23	6.52	12.19	0.04	3.04

The components of \mathcal{R}_0 (the reproduction number associated with each infectious compartment) can also be extracted.

```
knitr::kable(round(t(get_R0(params1, components=TRUE)),2))
```

37

asymptomatic	pre-symptomatic	mild	severe
1.56	0.33	4.46	0.17

38 **Initial conditions** The initial state must also be set, but it is sufficient to specify the
 39 parameter set (a `params_pansim` object), in which case the population size and initially ex-
 40 posed population will be taken from the parameters (in this case all non-exposed individuals
 41 are assumed to be susceptible).

```
state1 <- make_state(params=params1)
```

42 **Start and end dates** Dates on which the simulation starts and ends must be stated. If
 43 there are no observations that you are aiming to match, then these dates are arbitrary and
 44 only the length of time matters.

```
sdate <- "2020-Feb-10"  
edate <- "2020-Jun-1"
```

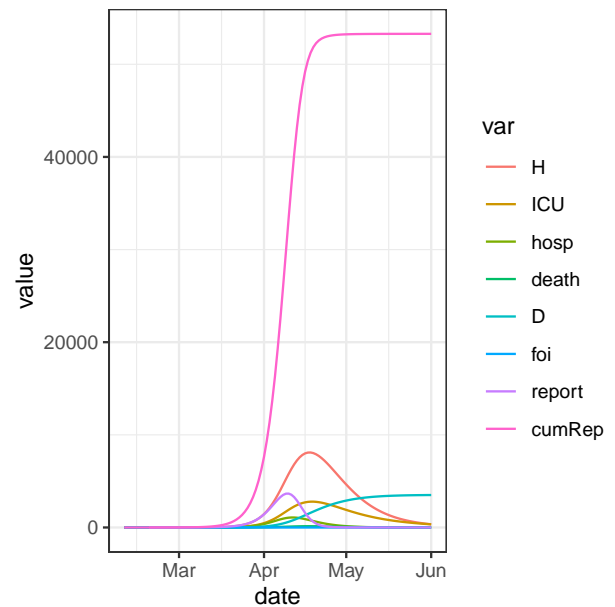
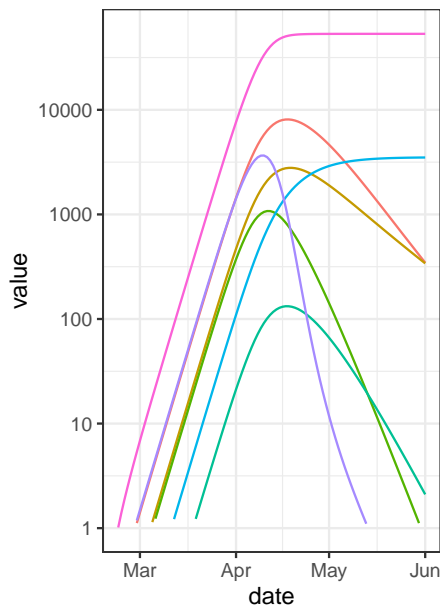
45 3 Running a simulation

46 A simple deterministic simulation is run as follows, and returns a `pansim` object. The
 47 `summary` method computes the times and magnitudes of peak demands on acute care (H)
 48 and intensive care (ICU), and the basic reproduction number \mathcal{R}_0 .

```
res1 <- run_sim(params=params1, state=state1, start_date=sdate, end_date=edate)  
summary(res1)  
  
#> peak_ICU_date peak_ICU_val peak_H_date peak_H_val R0  
#> 1 2020-04-19 2787 2020-04-18 8094 6.518009
```

49 The `plot` method for `pansim` objects returns a `ggplot` object, optionally on a log scale.

```
plot_grid(plot(res1, log=TRUE), ## logarithmic  
           plot(res1)) ## linear  
  
#> Warning: Removed 11 row(s) containing missing values (geom_path).
```



50

51 3.1 Stochasticity

52 The effects of observation error are easy to explore with the `stoch` argument to `run_sim`.

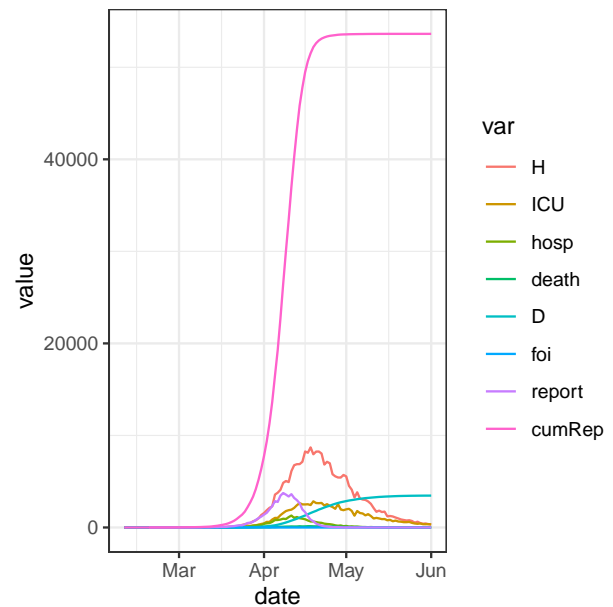
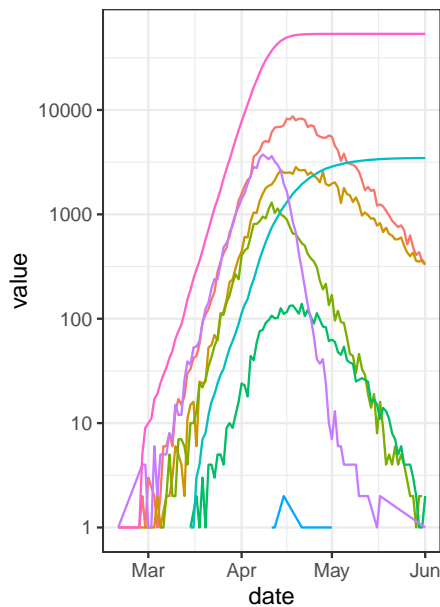
53 The `obs_disp` parameter is the dispersion parameter for a negative binomial.

```
set.seed(101)
params1obs <- update(params1, obs_disp=200)
res1obs <- run_sim(params1obs, state1, start_date=sdate, end_date=edate,
                   stoch=c(obs=TRUE, proc=FALSE))
summary(res1obs)

#>   peak_ICU_date peak_ICU_val peak_H_date peak_H_val      R0
#> 1 2020-04-19      2840 2020-04-18      8701 6.518009

plot_grid(plot(res1obs, log=TRUE),
           plot(res1obs))

#> Warning: Removed 11 row(s) containing missing values (geom_path).
```



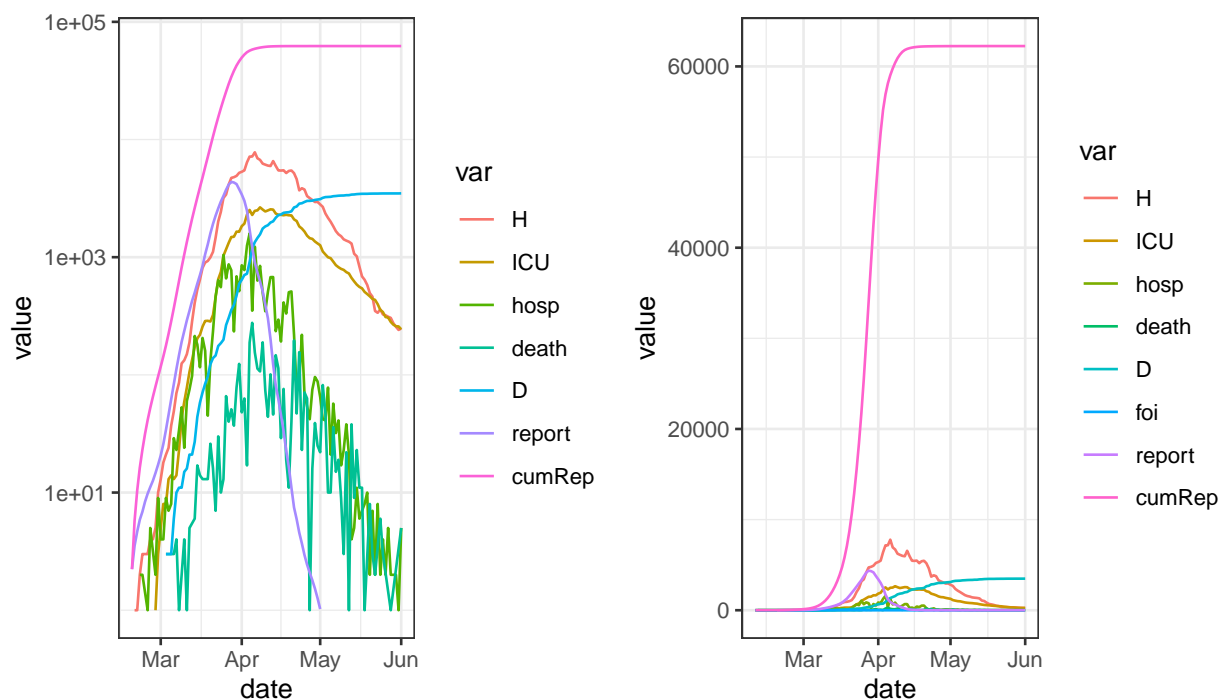
To simulate with process error, use `stoch=c(..., proc=TRUE)`. By default, this simulates only demographic stochasticity, which has little effect in a large epidemic.

```
params1proc <- update(params1,E0=200,proc_disp=0) ## demog stoch only
res1proc <- run_sim(params1proc, start_date=sdate, end_date=edate,
  stoch=c(obs=FALSE, proc=TRUE))
```

Making `proc_disp` positive simulates with additional process noise:

```
params1proc2 <- update(params1,E0=200, proc_disp=0.5, obs_disp=5)
res1proc2 <- run_sim(params1proc2, start_date=sdate, end_date=edate,
  stoch=c(obs=FALSE, proc=TRUE))
plot_grid(plot(res1proc2, log=TRUE), plot(res1proc2))

#> Warning: Removed 11 row(s) containing missing values (geom_path).
```



58

59 **Technical note.** Demographic noise is included by calculating probabilities from the rates
60 and then drawing a multinomial sample to determine how many individuals move from one
61 compartment to each of the others. With pure demographic noise, the CV is very small
62 with only ~ 1000 individuals moving among compartments. Process dispersion (`proc_disp`;
63 “overdispersed demographic stochasticity”) is implemented using `pomp::reulermultinom`,
64 which adds gamma white noise to the event rates. For some discussion of this, see p. 274 and
65 Appendix A of the “plug-and-play” paper by He *et al.* (2010, *J. R. Soc. Interface* **7**, 271–
66 283, doi:10.1098/rsif.2009.0151. [DE: *The intensity of the gamma white noise process*
67 *(proc_disp) has units (cf. σ_{SE} in He et al.); it would be easier to think about the coefficient*
68 *of variation (CV) rather than standard deviation (sd).]*

69 [DE: *Notes scribbled from discussion with BB: To get CIs on a forecast, we could hack*
70 *by adjusting proc_disp until getting CIs that are plausibly wide; estimating this number is*
71 *a can of worms. A slightly more principaled way to decide on that number: fit params, then*
72 *run sims with different combinations of obs and proc noise that yield noise like in the data:*
73 *then infer how observed noise is divided btw proc and measurement error.]*

74 [DE: *DC commented on 19 Apr 2020 (‘MP updates’ thread): “5/ I have had the same*
75 *question for a while regarding noise amplitude... I usually look at the variance of the data*
76 *as a guidance, but never did anything formal. 6/ I often find myself starting with MCMC,*
77 *just to give it up for ABC or something else a few days/weeks down the road because I*
78 *end up spending way too much time in trying to fix more or less technical issues regarding*
79 *convergence (I use Stan nearly all the time, maybe that’s why...).”]*

3.2 Time-dependent transmission rate

Implementing known changes in transmission rate (e.g., resulting from social distancing measures) is straightforward via the `time_pars` argument. The following reduces β_0 (and hence \mathcal{R}_0) to 50% of its original value on 10 March 2020, and to 10% of its original value on 25 March 2020.

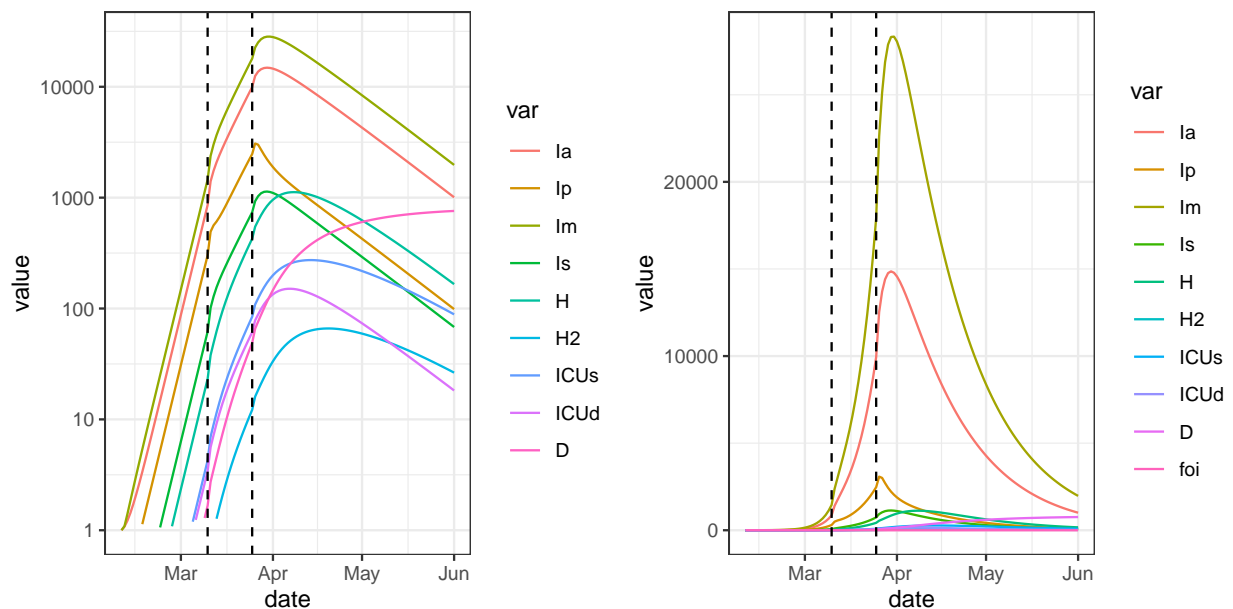
Setting `ndt=20` forces 20 intermediate time steps to occur between each saved step. (Try it with `ndt=1` to see why this is a good idea.)

Setting `condense=FALSE` retains all variables in the output, rather than collapsing into a single *I* class *etc.*

```
time_pars <- data.frame(Date=c("2020-Mar-10","2020-Mar-25"),
                        Symbol=c("beta0","beta0"),
                        Relative_value=c(0.5,0.1))
restimedep <- run_sim(params1,state1,start_date=sdate,end_date=edate,
                     params_timevar=time_pars,ndt=20, condense=FALSE)
summary(restimedep)

#>   peak_ICU_date peak_ICU_val peak_H_date peak_H_val      R0
#> 1 2020-04-10      415 2020-04-08      1175 6.518009

plot_grid(plot(restimedep, log=TRUE, condense=FALSE),
          plot(restimedep, condense=FALSE))
```



4 Changing parameters

Some parameters you might wish to change are not directly available in the parameter file. Instead, you can adjust them using `fix_pars()`. For example, if you would like to change

93 the default value of \mathcal{R}_0 implied in the parameter list `params1` you can do the following.

```
print(summary(params1))

#>           r0           R0           Gbar      CFR_gen      dbl_time
#> 0.2278149  6.5180089 12.1897402  0.0352000  3.0425898

## Change R0 to 2
newparams1 <- fix_pars(params1, target=c(R0=2))
print(summary(newparams1))

#>           r0           R0           Gbar      CFR_gen      dbl_time
#> 0.06649208 2.00002038 12.18974018  0.03520000 10.42450796
```

94 *[DE: See refactor.Rmd for functions not yet described here.]*

95 5 Calibration

96 In a typical epidemic forecasting application, we have imperfect information about the pa-
97 rameters and a time series of reported events (e.g., cases, hospitalizations, deaths, *etc.*). Our
98 goal is to predict the future course of the outbreak, and to determine how it will differ under
99 various intervention scenarios.

100 The natural approach is to find a set of parameters that lies within the estimated con-
101 straints and best fits the observed part of the epidemic. This is referred to as “calibrating”
102 the model to the data.

103 Unsurprisingly, there is a function `calibrate()` for doing just this.

104 Imagine that the simulated data saved in `res1obs` were the observed data to which want
105 to fit the model. We can calibrate to these data as follows.

106 Note that `calibrate()` requires the data come in “long form”, which means that for
107 each date on which we have data, there are separate rows for each type of data (report,
108 death, hospitalization, *etc.*). This is in contrast to “wide form”, for which there is one row
109 for each date, and separate columns for each observed variable.

110 *[DE: I am not supressing warnings so users aren’t alarmed when they see these warnings*
111 *themselves. We should, of course, consider revisions that either avoid warnings or explain*
112 *clearly to the user how to resolve them.]*

```
library(dplyr)
```

```

## pull out only the reported cases and convert to long form:
report_data <- (res1obs
  %>% mutate(value=round(report), var="report")
  %>% select(date, value, var)
  %>% na.omit()
)
head(report_data)

#>           date value    var
#> 10 2020-02-19      0 report
#> 11 2020-02-20      1 report
#> 12 2020-02-21      0 report
#> 13 2020-02-22      0 report
#> 14 2020-02-23      0 report
#> 15 2020-02-24      0 report

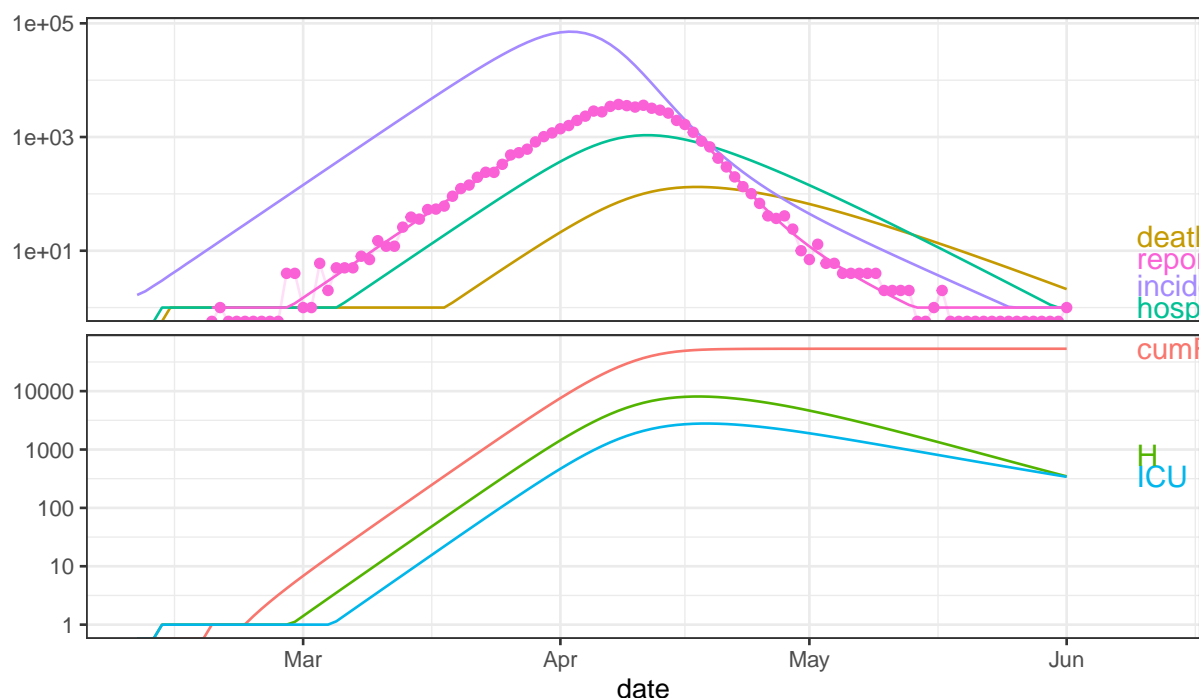
## beta0 is the only parameter we're going to optimize:
opt_pars <- list(params = c(beta0=0.1))
## fit beta0 based on the report data:
fitted.mod <- calibrate(
  data = report_data
, start_date = sdate
  ## skip breaks that are present by default:
, time_args = list(break_dates = NULL)
, base_params = params1obs
, opt_pars = opt_pars
  ##, debug_plot = TRUE # instructive plotting during optimization
)

#> Warning in optim(par = c(params.beta0 = 0.1), fn = function (p) : one-dimensional
optimization by Nelder-Mead is unreliable:
#> use "Brent" or optimize() directly

## plot the resulting fit
plot(fitted.mod, data=report_data)

#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Removed 11 row(s) containing missing values (geom_path).
#> Warning: Removed 11 rows containing missing values (geom_dl).

```



113

```
## spit out fitted parameters (in this case, just beta0)
coef(fitted.mod, "fitted")

#> $params
#>    beta0
#> 0.999375
```

114 That worked well, given that the value of `beta0` used for the simulation was 1. You might
 115 want to try running the above interactive without commenting out “`debug_plot = TRUE`”.
 116 This will allow you to see the process of fitting the model to the data. Note, however, that
 117 this instructive visualization of the optimization process will slow down the optimization by
 118 an order of magnitude.

119 Let’s now try to fit the model to both reports and deaths. It is easiest to create the
 120 required long-form data frame using the `pivot_longer` function in the `tidyr` package.

```
library(tidyr)
```

```

report_death_data <- (res1obs
  %>% select(date, report, death)
  %>% pivot_longer(names_to = "var", -date)
  %>% mutate(value=round(value))
  %>% na.omit()
)
head(report_death_data, n=12)

#> # A tibble: 12 x 3
#>   date      var    value
#>   <date>    <chr>  <dbl>
#> 1 2020-02-11 death     0
#> 2 2020-02-12 death     0
#> 3 2020-02-13 death     0
#> 4 2020-02-14 death     0
#> 5 2020-02-15 death     0
#> 6 2020-02-16 death     0
#> 7 2020-02-17 death     0
#> 8 2020-02-18 death     0
#> 9 2020-02-19 report     0
#> 10 2020-02-19 death     0
#> 11 2020-02-20 report     1
#> 12 2020-02-20 death     0

```

121

Now let's fit to both reports and deaths.

```
## beta0 is the only parameter we're going to optimize:
```

```

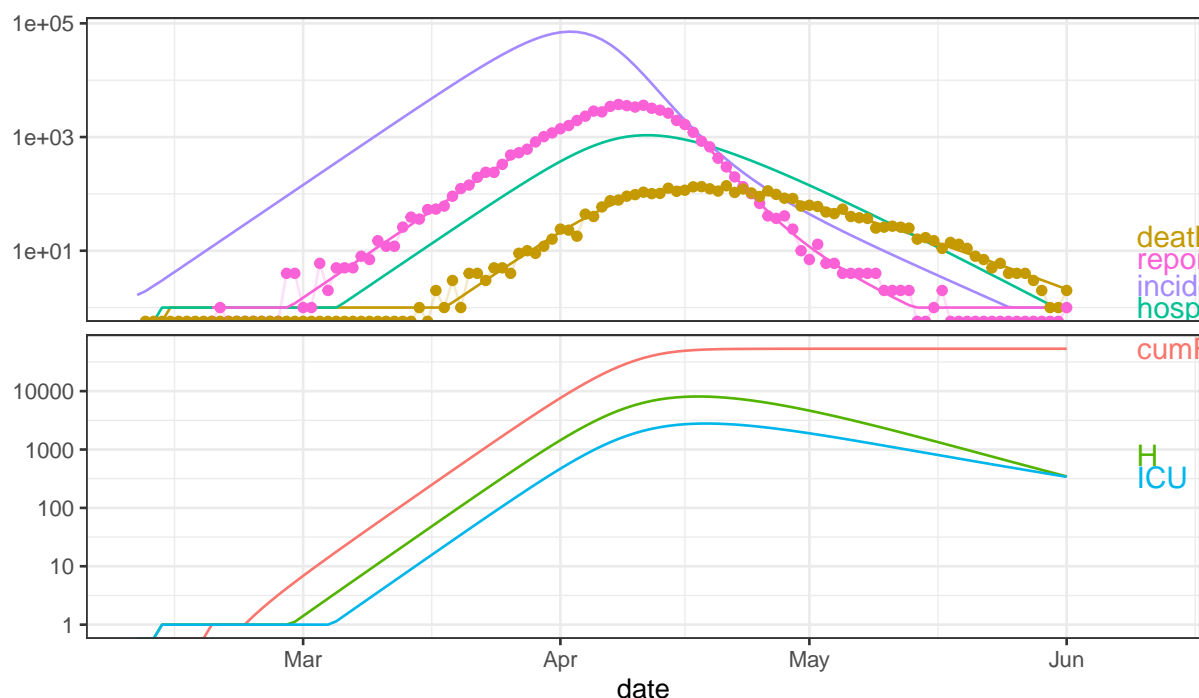
opt_pars <- list(params = c(beta0=0.1))
fitted.mod <- calibrate(
  data = report_death_data
, start_date = sdate
  ## skip breaks that are present by default:
, time_args = list(break_dates = NULL)
, base_params = params1obs
, opt_pars = opt_pars
  ##, debug_plot = TRUE # instructive plotting during optimization
)

#> Warning in dnbinom(value, mu = pred, size = nb_disp, log = TRUE): NaNs produced
#> Warning in optim(par = c(params.beta0 = 0.1), fn = function (p) : one-dimensional
optimization by Nelder-Mead is unreliable:
#> use "Brent" or optimize() directly
#> Warning in dnbinom(value, mu = pred, size = nb_disp, log = TRUE): NaNs produced
#> Warning in dnbinom(value, mu = pred, size = nb_disp, log = TRUE): NaNs produced

plot(fitted.mod, data=report_death_data)

#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Removed 11 row(s) containing missing values (geom_path).
#> Warning: Removed 11 rows containing missing values (geom_dl).

```



122

```
coef(fitted.mod, "fitted") # spit out fitted parameters

#> $params
#> beta0
#>      1

summary(coef(fitted.mod))

#>           r0           R0          Gbar      CFR_gen    dbl_time
#> 0.2278149  6.5180089 12.1897402  0.0352000  3.0425898
```

123 Note that in the summary at the end, `r0` refers to the initial exponential growth rate from the
 124 fitted model. This provides an alternative to the `epigrowthfit` package for fitting epidemic
 125 growth rates.

126 Finally, consider the case where we have both observation and process noise. Fitting to
 127 these data won't do as well, because `calibrate()` does not have a way of fitting to process
 128 noise. Consequently, the quality of our fit can be expected to be worse. Of course, real data
 129 always contain process noise...

```
report_data <- (res1proc2
```

```

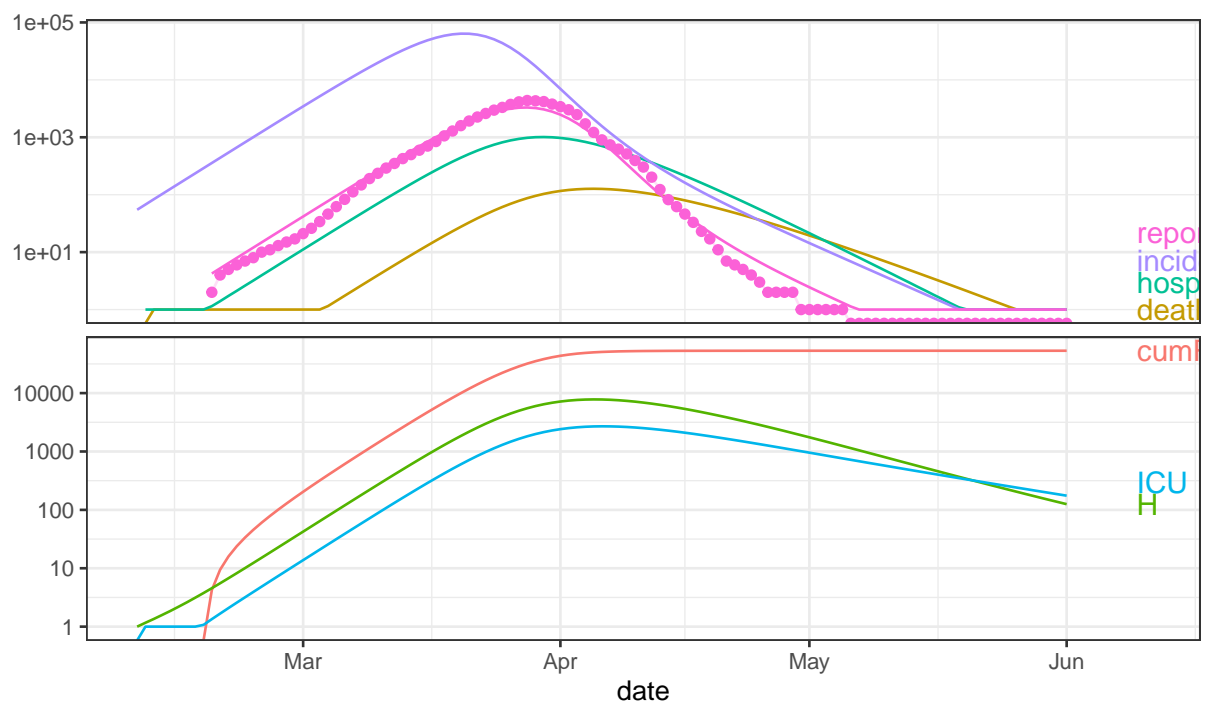
  %>% mutate(value=round(report), var="report")
  %>% select(date, value, var)
  %>% na.omit()
)
## beta0 is the only parameter we're going to optimize:
opt_pars <- list(params = c(beta0=0.1))
fitted.mod <- calibrate(
  data = report_data
, start_date = sdate
  ## skip breaks that are present by default:
, time_args = list(break_dates = NULL)
, base_params = params1proc2
, opt_pars = opt_pars
  ##, debug_plot = TRUE # instructive plotting during optimization
)

#> Warning in optim(par = c(params.beta0 = 0.1), fn = function (p) : one-dimensional
optimization by Nelder-Mead is unreliable:
#> use "Brent" or optimize() directly

plot(fitted.mod, data=report_data)

#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Transformation introduced infinite values in continuous y-axis
#> Warning: Removed 11 row(s) containing missing values (geom_path).
#> Warning: Removed 11 rows containing missing values (geom_dl).

```



130

```
coef(fitted.mod, "fitted") # spit out fitted parameters

#> $params
#> beta0
#> 0.8875

summary(coef(fitted.mod, "all"))

#>          r0          R0          Gbar      CFR_gen      dbl_time
#> 0.2082162  5.7847329 12.1897402  0.0352000  3.3289781
```