Getting started with the fitode package

Sang Woo Park

October 26, 2022

Contents

1 Introduction

`fitode` is an R package for fitting ordinary differential equations (ODE) using Maximum Likelihood or Bayesian Markov Chain Monte Carlo (MCMC). It relies on symbolic differentiation features of the `Deriv` package to solve the sensitivity equations so that gradient-based optimization algorithms can be used.

- response distributions: Gamma, Gaussian, Poisson, and negative binomial (NB1 and NB2 parameterization)
- link functions on model parameters: log, logit, and identity
- fitting multiple states to multivariate time series
- prior/penalization: Beta, Gamma, and Gaussian distributions
- confidence intervals on parameters and their transformations via delta method, profiling, and importance sampling

In order to construct a model in `fitode` you need to:

- specify the gradients using formula notation (e.g., $dX/dt = f(X)$ is expressed as `X ~ f(X)`)
- specify the observation process using formula notation (e.g., `Xobs ~ dnorm(mean=X, sd=sigma)`
- specify the initial conditions using formula notation
- specify the parameters of the model
- specify the link functions (log-link is the default)

To fit a model, you need to:

- specify the data (as well as the time column)
- specify the starting values for optimization or MCMC
- optionally specify fixed parameters
- optionally specify prior distributions (or penalizations); not specifying prior distribution in MCMC will result in improper priors on link scales

This document was generated using R version 4.1.2 (2021-11-01) and package versions:

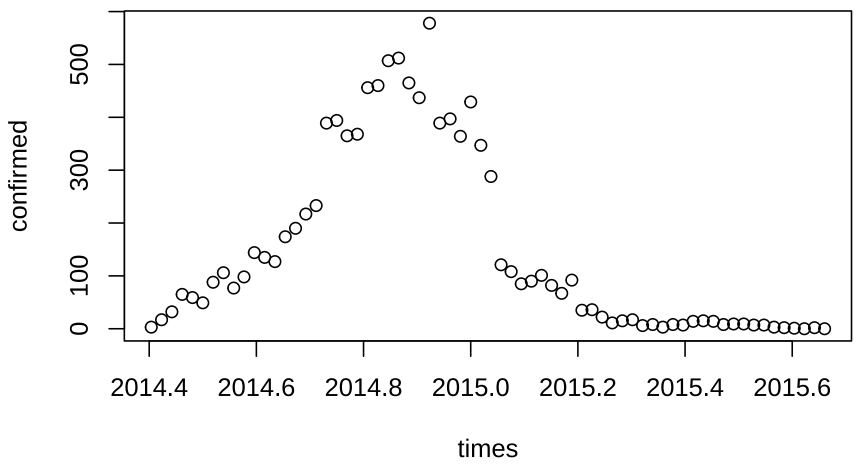
```
##   Deriv   bbmle deSolve  fitode ggplot2
##   4.1.3  1.0.24    1.30   0.1.1   3.3.6
```

2 Basic fitting - estimating epidemic growth rates

2.1 Data

Here, we study a time series of confirmed cases of Ebola during the 2014 outbreak in Sierra Leone to characterize epidemic growth patterns. Once you load `fitode`, the data set (`SierraLeone2014`) will be automatically loaded into the global environment.

```
library(ggplot2); theme_set(theme_bw())
library(fitode)
plot(SierraLeone2014)
```



2.2 Exponential growth model

Exponential growth is one of the simplest models we can use to characterize the initial spread of a disease:

$$\frac{dX}{dt} = rX. \tag{1}$$

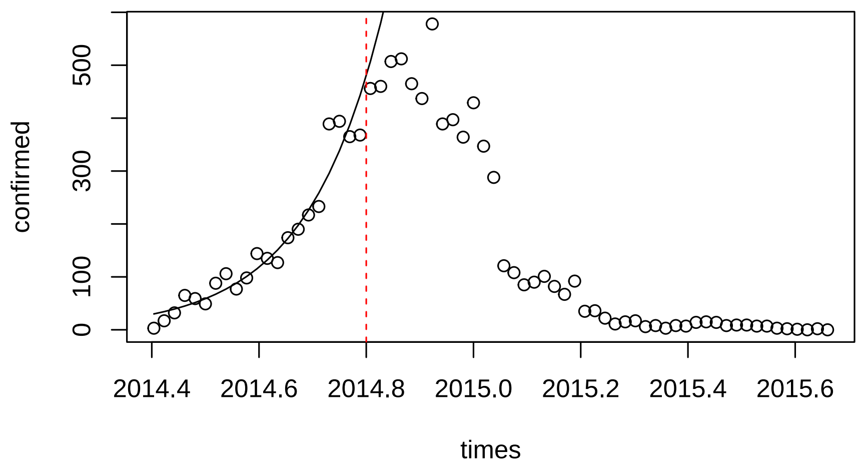
This model is parameterized by the initial growth rate r and the initial value $X(0)$. Variable X describes the dynamics of *mean* confirmed cases; we will assume that the observed number of confirmed cases at time t follows a Poisson error distribution with mean $X(t)$. This model can be constructed in `fitode` as follows:

```
exp_model <- odemodel(
    name="exponential",
    model=list(
        X ~ r * X
    ),
    observation=list(
        confirmed ~ dpois(lambda=X)
    ),
    initial=list(
        X ~ X0
    ),
    par=c("r", "X0")
)
```

Note that the name(s) of the observed variable(s) (here, `confirmed`) must be different from the name(s) of the state variable(s) (here, `X`).

In order to fit this model to the data, we have to specify starting parameters for the optimization. To do so, we can simulate the model for various parameters and try to find a reasonable parameter set by eye. For example, here is a parameter set found by trial and error:

```
start <- c(r=7, X0=30)
ss <- simulate(exp_model, parms=start, times=SierraLeone2014$times)
plot(SierraLeone2014)
lines(X ~ times, data=ss)
abline(v=2014.8, col="red", lty=2)
```



Here, we used the `simulate` function to simulate the model. It requires a parameter set (`parms` argument) and a time vector (`times` argument) to run. It returns a deterministic ODE solution for each state variable as well as stochastic simulated observations based on the ODE solution; we will ignore the simulated observations for now.

The data does not exhibit exponential growth forever. In order to fit the exponential model, we have to determine a fitting window. Here we will fit the model from the beginning of the epidemic to time 2014.8 (red dashed line in the previous figure).

```
exp_fit <- fitode(
    model=exp_model,
    data=subset(SierraLeone2014, times <= 2014.8),
    start=start
)

## Fitting ode ...
## Computing vcov on the original scale ...
```

The estimated parameters are very close to our initial guess:

```
exp_fit

## Model: exponential
##
## Observations:
## confirmed ~ dpois(lambda = X)
##
## Coefficients:
##         r        X0
##  6.993036 30.249604
##
## Log-Likelihood:-140.07
##
## link: r = log; X0 = log
```

Since the exponential ODE has a simple closed-form analytical solution, we could also have used MLE directly in this case:

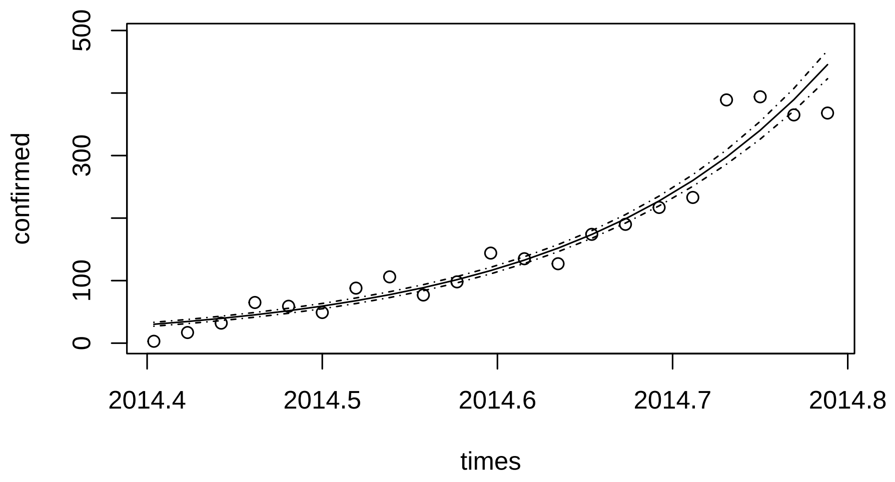
```
mlefit <- bbmle::mle2(confirmed ~ dpois(X0*exp(r*(times-times[1]))),
                      data=subset(SierraLeone2014, times <= 2014.8),
                      start=as.list(start))
coef(mlefit)

##         r        X0
##  6.993011 30.249836
```

We can quantify the uncertainty in the parameters by using `confint`:

```
confint(exp_fit)

##      estimate      2.5 %    97.5 %
## r    6.993036  6.652687  7.350797
## X0 30.249604 27.308713 33.507203
```

By default, `confint` will calculate the confidence intervals using the delta method. bmb: *are these 'delta method' or Wald CIs??* We diagnose the fit by using the `plot` function (the `level=0.95` argument specifies that 95% confidence intervals should be drawn):

```
plot(exp_fit, level=0.95)
```



The confidence intervals on our predictions are suspiciously narrow, probably because of our choice of the error function. The Poisson distribution assumes that variance of the residuals is equal to the mean (i.e., the fitted value). Instead, we can use a negative binomial distribution, which assumes that variance is a quadratic function of the mean. Then, we have to estimate an extra parameter (`size` argument of the `dnbinom`) to account for overdispersion. We use the `update` function to adjust only these particular aspects of the model, leaving the gradient specification the same:

```
exp_fit_nbinom <- update(
    exp_fit,
    observation=list(
        confirmed ~ dnbinom(mu=X, size=phi)
    ),
    par=c("r", "X0", "phi"),
    start=c(start, phi=10)
)

## Fitting ode ...
## Computing vcov on the original scale ...
```

Note that we need to specify a starting value for the overdispersion parameter as well.

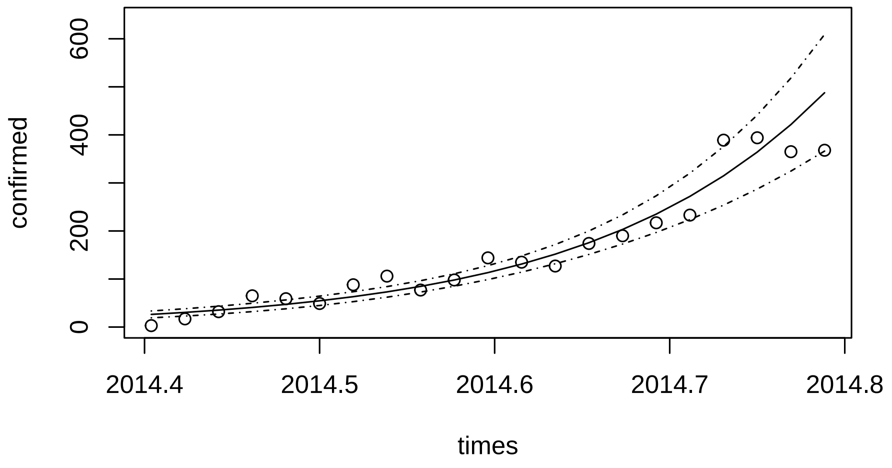
Alternatively, we can update the `odemodel` object and refit the model:

```
exp_model_nbinom <-
  update(exp_model,
         name="exponential (nbinom)",
         observation=list(
           confirmed ~ dnbinom(mu=X, size=phi)
         ),
         par=c("r", "X0", "phi")
         )

exp_fit_nbinom2 <- fitode(
    model=exp_model_nbinom,
    data=SierraLeone2014[SierraLeone2014$times <+ 2014.8,],
    start=c(start, phi=10)
)

## Fitting ode ...
## Computing vcov on the original scale ...
```

Both approaches give the same results.

We can plot this fit:

```
plot(exp_fit_nbinom, level=0.95)
```



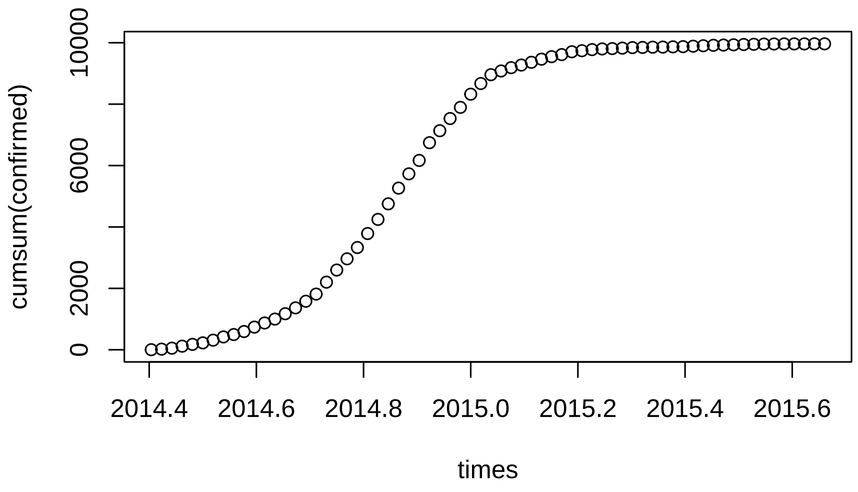
Our uncertainty is now more reasonable. This change widens the confidence intervals on parameters as well:

```
confint(exp_fit_nbinom)

##      estimate     2.5 %    97.5 %
## r    7.589514  6.500229  8.861336
## X0  26.326482 20.089257 34.500213
## phi 12.903678  5.266959 31.613098
```

2.3 Logistic growth model

Exponential growth model accounts for only the initial portion of the observed data. Instead, we might want to try to model the entire time series. Note that the cumulative number of cases saturates over time:

```
plot(cumsum(confirmed) ~ times, data=SierraLeone2014)
```



We can use a logistic model to describe this saturating pattern:

$$\frac{dX}{dt} = rX\left(1 - \frac{X}{K}\right). \tag{2}$$

While we can fit X directly to cumulative number of cases, it can lead to overly confident results due to accumulation of observation error (King et al., 2015). Instead, we can use *interval counts* to model the true number of cases: $X(t) - X(t - \Delta t)$, where Δt is the reporting time step. This is done by using the `diffnames` argument

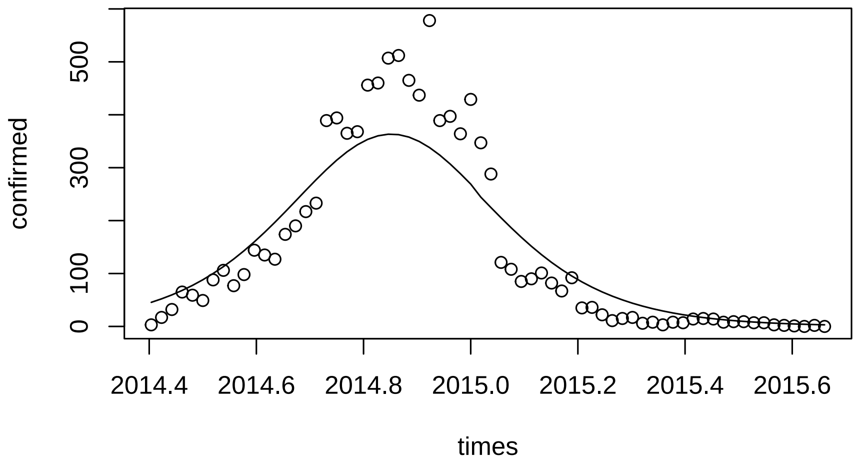
```
logistic_model <- update(
  exp_model_nbinom,
  name="logistic (nbinom)",
  model=list(
    X ~ r * X * (1 - X/K)
  ),
  diffnames="X",
  par=c("r", "X0", "K", "phi")
)
```

In this case, we need to modify the data set by adding an extra `NA` observation before the first observation; this allows `fitode` to take the interval difference and still end up with the same number of observations as the time series.

```
SierraLeone2014b <- rbind(
    c(times=SierraLeone2014$times[1] -
          diff(SierraLeone2014$times)[1], confirmed=NA),
    SierraLeone2014
)
```

Again, we can try to find a reasonable parameter set by trial and error:

```
start_logistic <-
    c(coef(exp_fit_nbinom), K=sum(SierraLeone2014$confirmed))
## need to use a different value for X0
start_logistic[["X0"]] <- 300
ss_logistic <- simulate(
    logistic_model,
    parms=start_logistic,
    times=SierraLeone2014b$times
)

plot(SierraLeone2014)
lines(X~times, data=ss_logistic)
```



and fit the model:

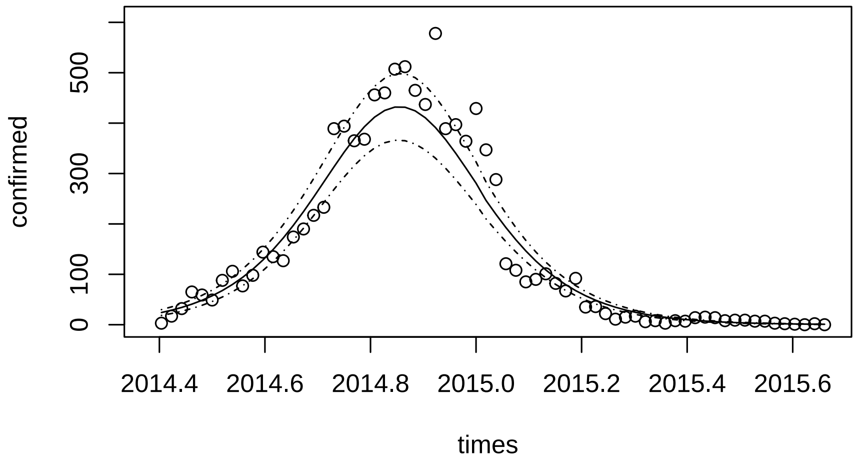
```
logistic_fit <- fitode(
    logistic_model,
    data=SierraLeone2014b,
    start=start_logistic
)

## Fitting ode ...
## Computing vcov on the original scale ...
```

In this case, we get a much higher growth rate estimate:

```
confint(logistic_fit)

##         estimate       2.5 %       97.5 %
## r       9.404301    8.879291     9.960355
## X0    123.985064   93.098091   165.119348
## K    9574.456216 8526.119846 10751.691682
## phi     7.814186    4.669271    13.077309
```

Plot:

```
plot(logistic_fit, level=0.95)
```



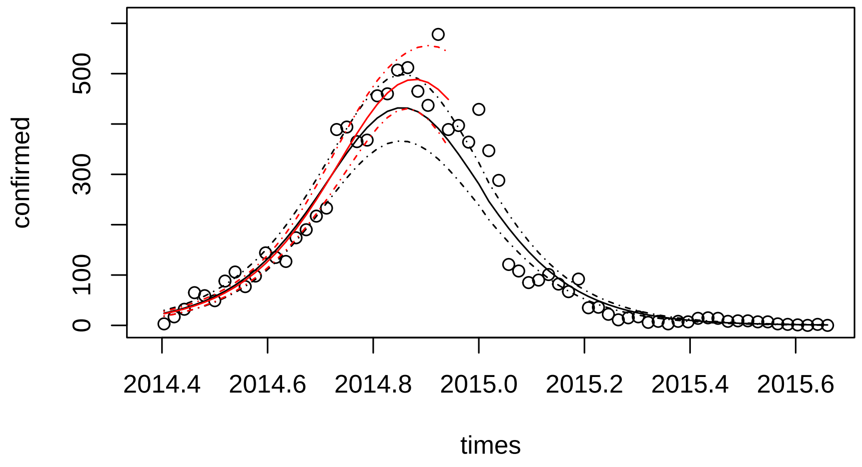
There is a clear bias in our fit; the estimated trajectory underestimates the peak of the epidemic. This is likely to affect our parameter estimates.

We can be smarter about our choices of fitting window. Instead of using the entire time series, we can fit the logistic model from the beginning of the epidemic to the next observation after the peak (Ma et al., 2014).

```
ma_begin <- 1
ma_end <- which.max(SierraLeone2014b$confirmed) + 1

logistic_fit_ma <- update(
    logistic_fit,
    data=SierraLeone2014b[ma_begin:ma_end,]
)

## Fitting ode ...
## Computing vcov on the original scale ...
```

We get a much better fit:

```
plot(logistic_fit, level=0.95)
plot(logistic_fit_ma, level=0.95, add=TRUE, col.traj="red", col.conf="red")
```



We get slightly wider confidence intervals on the parameters because we're using less data:

```
confint(logistic_fit_ma)

##         estimate       2.5 %      97.5 %
## r        9.29878    8.324641    10.38691
## X0     119.49151   86.681357   164.72078
## K    10943.72524 9183.475211 13041.37263
## phi     29.19584   12.515092    68.10952
```

2.4 SIR model

The Susceptible-Infected-Recovered (SIR) model describes how disease spreads in a homogeneous population:

$$\begin{aligned}\frac{dS}{dt} &= -\beta S\frac{I}{N}\\ \frac{dI}{dt} &= \beta S\frac{I}{N} - \gamma I\\ \frac{dR}{dt} &= \gamma I\end{aligned} \tag{3}$$

We can assume that confirmed cases are put into control and are no longer infectious, thus effectively recovering from infection (He et al., 2009); in other words, we model cumulative number of confirmed cases with cumulative number of recovered cases (state variable R).

Again, we use interval counts by using `diffnames="R"`:

```
SIR_model <- odemodel(
    name="SIR (nbinom)",
    model=list(
        S ~ - beta * S * I/N,
        I ~ beta * S * I/N - gamma * I,
        R ~ gamma * I
    ),
    observation=list(
        confirmed ~ dnbinom(mu=R, size=phi)
    ),
    initial=list(
        S ~ N * (1 - i0),
        I ~ N * i0,
        R ~ 0
    ),
    diffnames="R",
    par=c("beta", "gamma", "N", "i0", "phi"),
    link=c(i0="logit")
)
```

For simplicity, we assumed that there are no recovered individuals at the beginning of the epidemic[1]. The initial conditions are given by

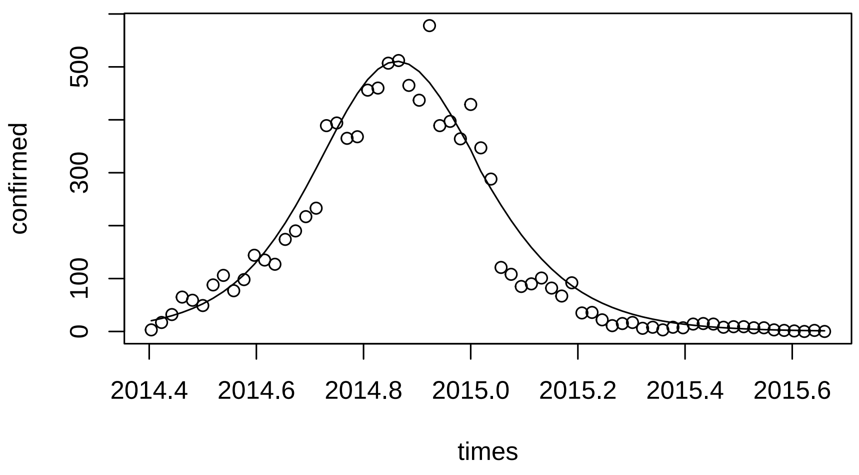
$$\begin{aligned}S(0) &= N(1 - i_0)\\ I(0) &= Ni_0\\ R(0) &= 0\end{aligned} \tag{4}$$

[1]Since these individuals would be completely uninvolved in the epidemic, and we are estimating the population size, we can make this assumption without any loss of generality

where i_0 is the initial proportion of infected individuals. Setting `link=c(i0="logit")` tells `fitode` that the parameter `i0` needs to be between 0 and 1. [2]

Searching for starting values:

```
SIR_start <- c(beta=70, gamma=60, N=40000, i0=0.0004, phi=6)

ss_SIR <- simulate(SIR_model,
    parms=SIR_start, times=SierraLeone2014b$times)

plot(SierraLeone2014)
lines(ss_SIR$times, ss_SIR$R)
```



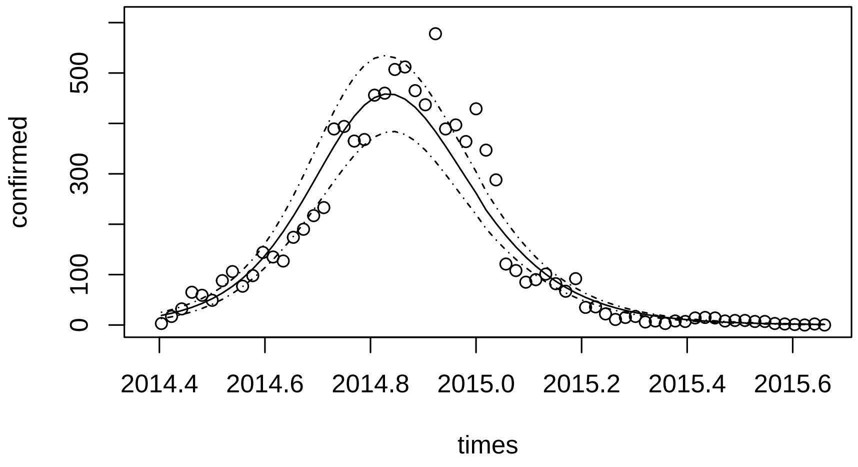
Fit:

```
SIR_fit <- fitode(
    SIR_model,
    data=SierraLeone2014b,
    start=SIR_start
)

## Fitting ode ...
## Computing vcov on the original scale ...
```

Plot:

[2]The *logit*, or log-odds, function (`qlogis()` in R), is the inverse of a logistic curve; it is a natural way to transform a value from the range [0,1] to $[-\infty, \infty]$.

```r
plot(SIR_fit, level=0.95)
```

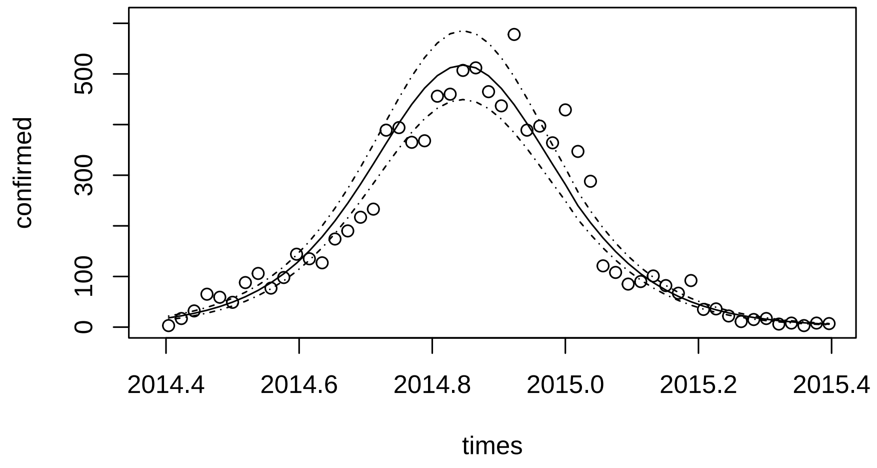


Again, the SIR model underestimates the peak.

This could be a problem with fitting window. When we get rid of the long tail in the time series, we get a much better fit:

```r
SIR_fit_b <- update(
    SIR_fit,
    data=SierraLeone2014b[SierraLeone2014b$times < 2015.4,]
)

## Fitting ode ...
## Computing vcov on the original scale ...
```

```r
plot(SIR_fit_b, level=0.95)
```



There are several ways we can get the confidence intervals on the growth rate ($r = \beta - \gamma$). By default, the package uses the delta method.

```
confint(SIR_fit_b, parm=list(r~beta-gamma))

##    estimate    2.5 %   97.5 %
## r 10.49462 9.919463 11.06978
```

We discuss other methods later.

Figure 1 compares the results of all of the methods we have tried.

3 Advanced fitting - multivariate time series

Data:

```
## FIXME: store these data locally
hare <- read.csv("https://raw.githubusercontent.com/stan-dev/example-models/master/knitr/lot
plot(Hare~Year, data=hare, type="l")
lines(Lynx~Year, data=hare, type="l", col=2)
```

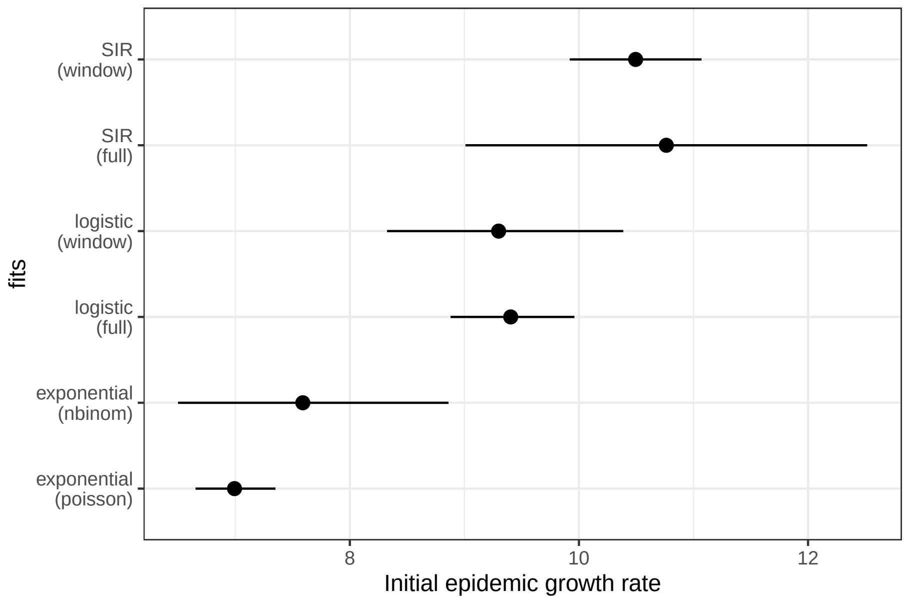
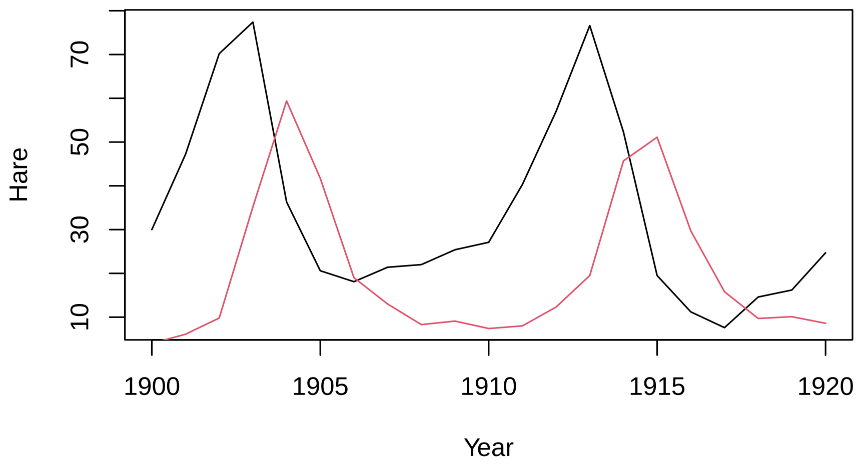


Figure 1: Comparison of growth rate estimates



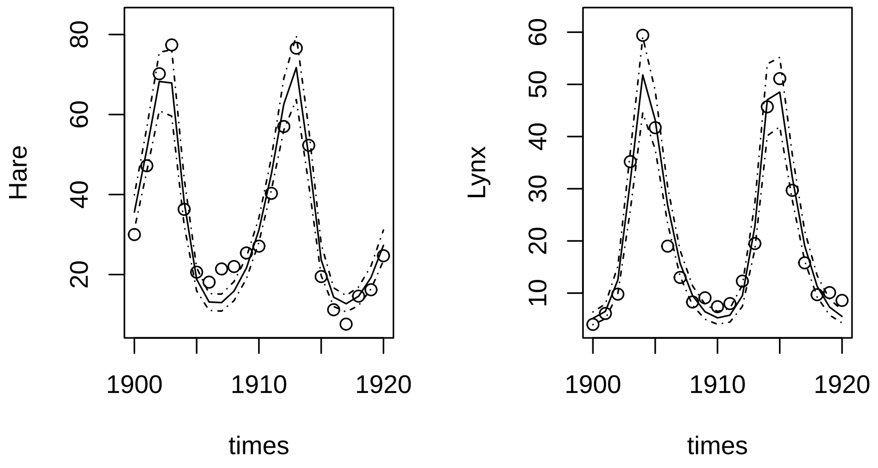
Lotka-Volterra model:

$$\begin{aligned}\frac{du}{dt} &= \alpha u - \beta uv \\ \frac{dv}{dt} &= \delta uv - \gamma v\end{aligned} \tag{5}$$

```r
lotka_model <- odemodel(
    name="Lotka Volterra model",
    model=list(
        u ~ alpha * u - beta * u * v,
        v ~ delta * u * v - gamma * v
    ),
    observation=list(
        Hare ~ dnbinom(mu=u, size=size1),
        Lynx ~ dnbinom(mu=v, size=size2)
    ),
    initial=list(
        u ~ u0,
        v ~ v0
    ),
    par=c("alpha", "beta", "delta", "gamma", "u0", "v0", "size1", "size2")
)
```

Fit with good starting values (estimated by someone else):

```r
harestart <- c(alpha=0.55, beta=0.028, delta=0.026, gamma=0.84, u0=30, v0=10,
               size1=1, size2=1)
harefit <- fitode(lotka_model, data=hare,
                  start=harestart,
                  tcol="Year")

## Fitting ode ...
## Computing vcov on the original scale ...

plot(harefit, level=0.95)
```



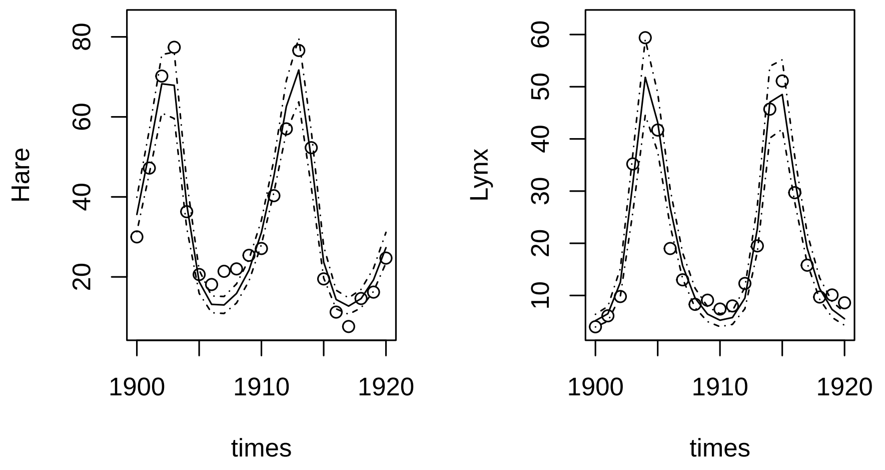
Esimates of size parameters are extremely large:

```r
coef(harefit)

##        alpha         beta        delta        gamma           u0           v0
## 5.047130e-01 2.479401e-02 2.518393e-02 8.582659e-01 3.560082e+01 5.174676e+00
##        size1        size2
## 1.831226e+05 5.365556e+07
```

This suggests that Poisson is actually good enough:

```r
## FIXME: we need this (fancy stuff with filling in all
## of the links as log) now because I'm checking links more carefully
## is there a way around this?
poisson_pars <- setdiff(lotka_model@par, c("size1", "size2"))
harefit_poisson <- update(
    harefit,
    observation=list(
        Hare ~ dpois(lambda=u),
        Lynx ~ dpois(lambda=v)
    ),
    link=setNames(rep("log",length(poisson_pars)),poisson_pars),
    par=poisson_pars
)

## Fitting ode ...
## Computing vcov on the original scale ...
```

```
plot(harefit_poisson, level=0.95)
```



Using `confint()` on the two models (with the default method, Wald approximation) shows that the confidence intervals on the parameters are nearly identical — except for the two negative binomial parameters which have extremely (ridiculously) wide confidence intervals, e.g. the 95% CI for the dispersion parameter on hares is $\{2.220446 \times 10^{-16}, 3.5505246 \times 10^{38}\}$. This problem occurs because the standard Wald approximation fails badly for these parameters.

We can get *lower* bounds on the confidence intervals for the dispersion parameters by using likelihood profiling. We have to work a little harder; we (1) manually set the parameter standard error to provide an initial scale for the profile (since the Wald estimate of the standard errors fails badly in this case) and (2) allow the profiling to proceed even if it discovers a fit that is slightly better (by up to 0.1 log-likelihood units) than the original fit.

```
confint(harefit,
        parm=c("size1","size2"),
        method="profile",
        std.err=1,
        tol.newmin=0.1)

##          estimate    2.5 % 97.5 %
## size1   183122.6 60.74175      NA
## size2 53655559.5 44.55821      NA
```

The results tell us that the upper 95% CIs are undefined (as would be expected if the model is not significantly better than Poisson), and the lower 95% CIs are ≈ 50.

References

He, D., E. L. Ionides, and A. A. King (2009). Plug-and-play inference for disease dynamics: measles in large and small populations as a case study. *Journal of the Royal Society Interface 7*(43), 271–283.

King, A. A., M. Domenech de Cellès, F. M. Magpantay, and P. Rohani (2015). Avoidable errors in the modelling of outbreaks of emerging pathogens, with special reference to Ebola. *Proceedings of the Royal Society B: Biological Sciences 282*(1806), 20150347.

Ma, J., J. Dushoff, B. M. Bolker, and D. J. Earn (2014). Estimating initial epidemic growth rates. *Bulletin of Mathematical Biology 76*(1), 245–260.