Basic ODE fitting

November 12, 2018

Contents

1 Preliminaries

Load packages:

```
library(fitode)
```

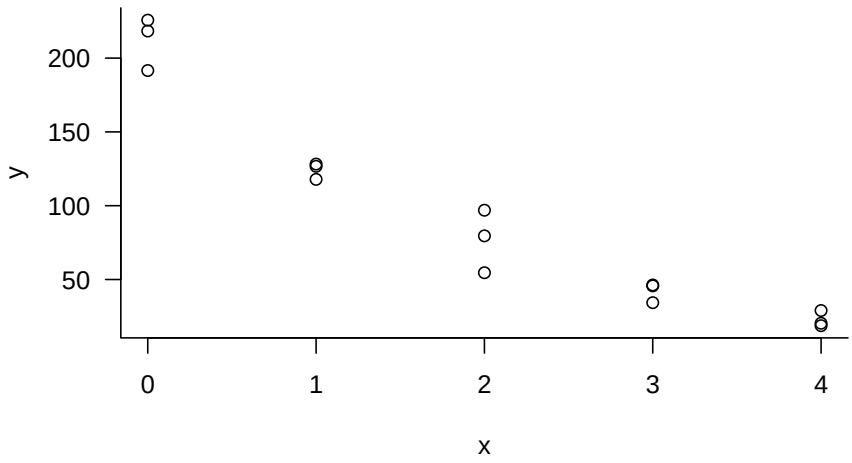
2 Basic fitting

2.1 Exponential decay model

Suppose we have a stock (compartment) that is losing a fixed *per capita* fraction per unit time, as in radioactive decay, financial depreciation, or a pure-death process.

```
set.seed(123)
true.m <- 0.5
true.A0 <- 200
true.sd <- 15

exp.data <- data.frame(
    x=rep(0:4, 3),
    y=rnorm(15, true.A0 * exp(-true.m * rep(0:4, 3)), sd=true.sd)
)
```

```
plot(exp.data)
```



The true dynamics can be modeled with the following equation:

$$\frac{dA}{dt} = -mA$$

We can translate this into a `fitode` model as follows:

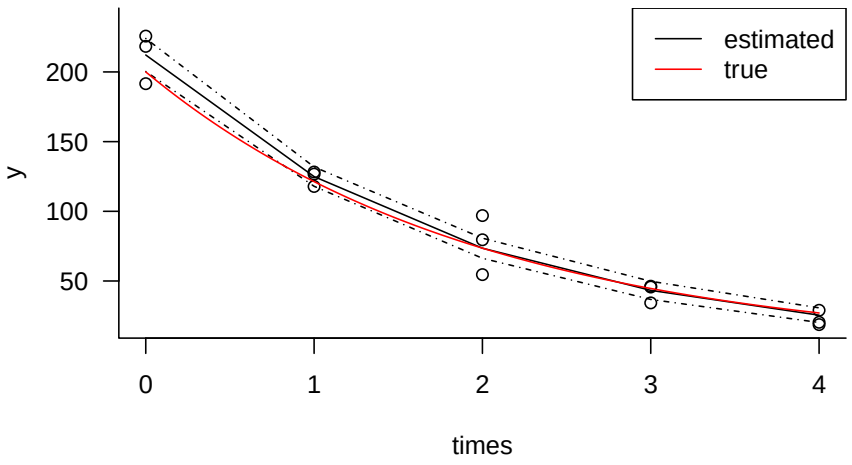
```
exp.model <- new("model.ode",
    name = "SI",
    model = list(
        A ~ -m * A
    ),
    observation = list(
        y ~ dnorm(mean=A, sd=sd)
    ),
    initial = list(
        A ~ A0
    ),
    par=c("m", "A0", "sd")
)
```

Then, we can fit the model:

```
exp.fit <- fitode(
    exp.model,
    exp.data,
    start=c(m=0.5, A0=200, sd=15),
    tcol="x"
)

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

To diagnose the fit, we can use the `plot` method for `fitode` objects. bmb: *we should be careful here: in base R the convention is that* `plot` *shows diagnostics (although of course that's different in* `deSolve`*, which plots trajectories). Do we want to set up different methods for plotting predictions vs trajectories, or should there be an option to* `plot`*?* By default, this will plot the estimated trajectory along with 95% confidence intervals, estimated via the delta method.

```
plot(exp.fit, level=0.95)
curve(true.A0 * exp(-true.m*x), add=TRUE, lty=1, col="red")
legend(
    x="topright",
    legend=c("estimated", "true"),
    col=c("black", "red"),
    lty=1
)
```



To obtain confidence intervals on the parameters, we can use `confint` method. There are three available methods for obtaining the confidence intervals: `delta`, `profile` and `wmvrnorm`. Due to computation speed, default option is `delta`. We will get into the details later. For now,

```
confint(exp.fit)
```

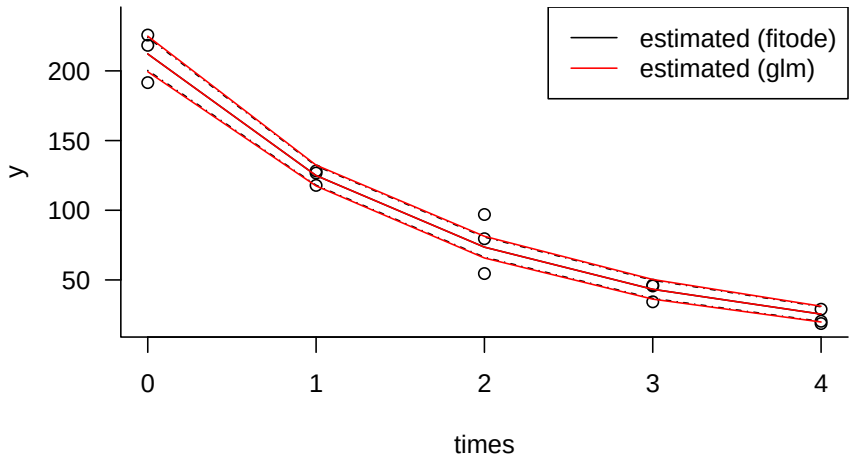
```
##       estimate       2.5 %      97.5 %
## m    0.5303757   0.4757616   0.5912593
## A0 212.1596895 200.6325622 224.3490954
## sd  11.0526632   7.7278640  15.8079080
```

Since this particular differential equation has a simple closed-form solution ($A(t) = A(0)\exp(-mt)$), we can use the `glm` function to fit the model as well (we need to use `glm` with the `gaussian` family and a log link, rather than `lm`, in order to fit an exponential curve with constant residual variance).

```
glm.fit <- glm(y~x,
    family=gaussian(link="log"),
    data=exp.data,
    start = c(intercept=log(200), x=-0.5))
```

Comparing the results:

```
glm.pred <- predict(glm.fit, data.frame(x=0:4), se.fit=TRUE, type="response")

glm.data <- data.frame(
    x=0:4,
    estimate=glm.pred$fit,
    lwr=glm.pred$fit-1.96 * glm.pred$se.fit,
    upr=glm.pred$fit+1.96 * glm.pred$se.fit
)

plot(exp.fit, level=0.95)
lines(glm.data$x, glm.data$estimate, col=2)
lines(glm.data$x, glm.data$lwr, col=2)
lines(glm.data$x, glm.data$upr, col=2)
legend(
    x="topright",
    legend=c("estimated (fitode)", "estimated (glm)"),
    col=c("black", "red"),
    lty=1
)
```



The estimated trajectories and their confidence intervals are essentially indistinguishable.

2.2 Chemical reaction - multiple state fitting

Now, consider the following chemical reaction:

$$A \rightarrow 3B$$

Then, we can write the governing differential equations as follows:

$$\frac{dA}{dt} = -kA$$
$$\frac{dB}{dt} = 3kB$$

Suppose we have observed values for both quantities (we assume their standard deviation is identical):

Taking a quick look at the simulation results:

```
head(reaction_data)
```

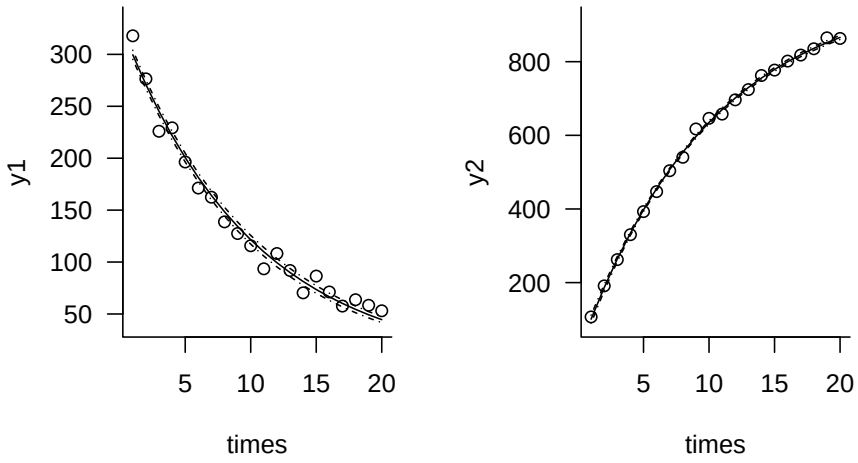
```
##   times       y1       y2
## 1     1 317.8691 106.8864
## 2     2 276.4298 191.1854
## 3     3 225.9531 262.5231
## 4     4 229.2591 330.2038
## 5     5 196.3682 392.9070
## 6     6 171.2810 447.1751
```

Here, `y1` measures quantity A and `y2` measures quantity B. bmb: *is there a reason not to use names that match the ODE definition?* Then, we can define the model:

```
reaction_model <- new("model.ode",
    name = "reaction",
    model = list(
        A ~ - k * A,
        B ~ 3 * k * A
    ),
    observation = list(
        y1 ~ dnorm(mean=A, sd=sd),
        y2 ~ dnorm(mean=B, sd=sd)
    ),
    initial = list(
        A~A0,
        B~B0
    ),
    par=c("k", "A0", "B0", "sd")
)
```

We can fit this using arbitrary starting conditions.

```
reaction_fit <- fitode(
    reaction_model,
    reaction_data,
    start=c(k=0.1, A0=300, B0=10, sd=10)
)

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

plot(reaction_fit, level=0.95)
```

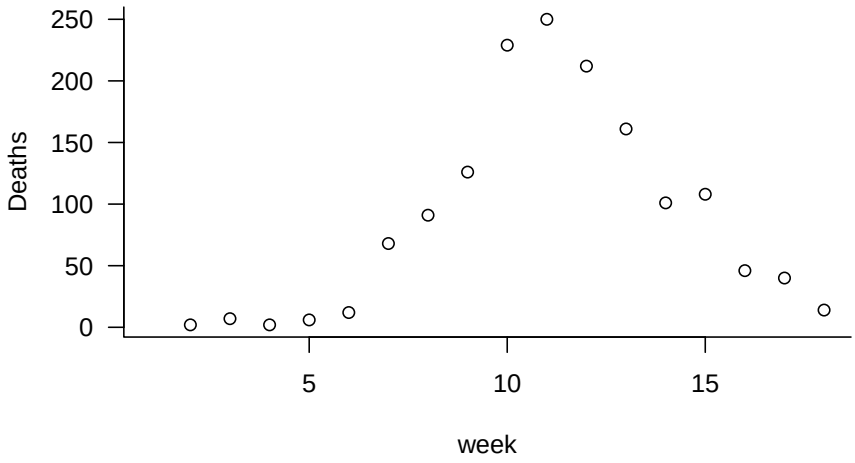


Confidence intervals...

```
confint(reaction_fit)

##      estimate        2.5 %      97.5 %
## k    0.100314   0.09714102   0.1035907
## A0 299.581692 295.20809184 304.0200894
## B0 100.734330  90.39960933 112.2505441
## sd   9.151578   7.35070035  11.3936588
```

2.3 Fitting SIR model

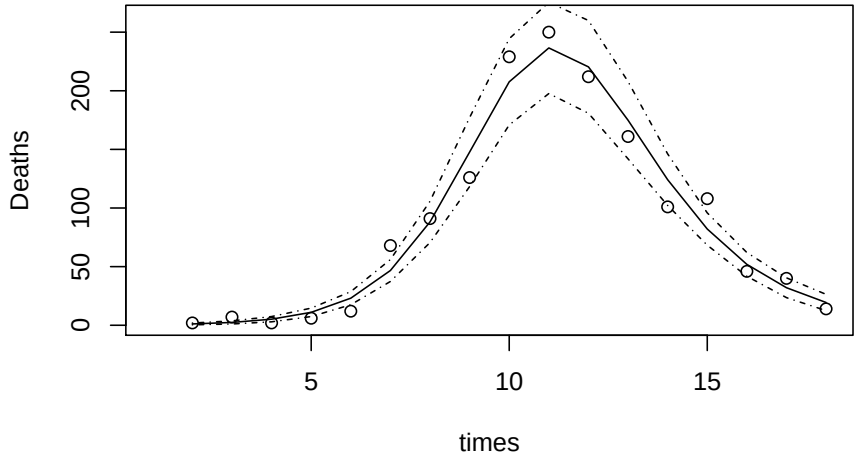
```
harbin <- fitsir::harbin
harbin2 <- rbind(data.frame(week=1, Deaths=NA), harbin)

plot(harbin2)
```



We need to add an `NA` observation to make this work...

```
SI_model_c <- new("model.ode",
    name = "SI",
    model = list(
        S ~ - beta*S*I/N,
        I ~ beta*S*I/N - gamma*I,
        cDeath ~ gamma*I
    ),
    observation = list(
        Deaths ~ dnbinom(mu=cDeath, size=size)
    ),
    initial = list(
        S ~ N * (1 - i0),
        I ~ N * i0,
        cDeath ~ 0
    ),
    diffnames="cDeath",
    par=c("beta", "gamma", "N", "i0", "size")
)

start <- c(beta=2, gamma=1, N=20000, i0=1e-5, size=10)

sirfit <- fitode(
    SI_model_c,
    harbin2,
```

```
    start=start,
    link = list(
        beta="log",
        gamma="log",
        N="log",
        i0="logit",
        size="log"
    ),
    tcol="week"
)

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

plot(sirfit, level=0.95)
```



Confidence intervals on various epidemiological quantities:

```
confint(sirfit,
        parm=list(
            R0~beta/gamma,
            r~beta-gamma
        ),
        method="wmvrnorm")

##    estimate     2.5 %   97.5 %
## R0 1.705258 1.4733837 2.132810
```

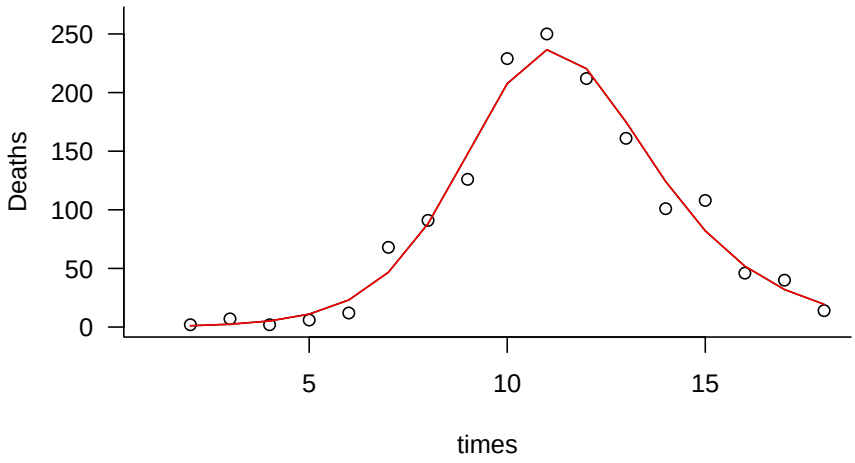
```
## r  0.772026 0.6942409 0.837602
```

Alternate parameterization:

```
SI_model_c2 <- Transform(
    SI_model_c,
    list(
        beta~(R0_1+1)*gamma
    ),
    par=c("R0_1", "gamma", "N", "i0", "size")
)

cc <- coef(sirfit)

start2 <- c(R0_1=unname(cc[1]/cc[2]-1), cc[-1])

sirfit2 <- fitode(
    SI_model_c2,
    harbin2,
    start=start2,
    link = list(
        R0_1="log",
        gamma="log",
        N="log",
        i0="logit",
        size="log"
    ),
    tcol="week"
)

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

Compare fits:

```
plot(sirfit)
plot(sirfit2, add=TRUE, col.traj="red", col.conf="red")
```



We get identical fits. The advantage of this parameterization is that we can obtain profile confidence intervals on R0 (it's a little slow...):

```
set.seed(101)
confint(sirfit2, "R0_1", method="profile") + 1

##      estimate    2.5 %   97.5 %
## R0_1 1.705259 1.000006 2.591917

confint(sirfit2, "R0_1", method="wmvrnorm") + 1

##      estimate    2.5 %   97.5 %
## R0_1 1.705259 1.238815 2.373596

confint(sirfit, parm=list(R0~beta/gamma))

##    estimate     2.5 %   97.5 %
## R0 1.705258 0.9361381 2.474379

confint(sirfit, parm=list(R0~beta/gamma), method="wmvrnorm")

##    estimate   2.5 %   97.5 %
## R0 1.705258 1.47908 2.147964
```

I'm not sure why performing `wmvrnorm` on the beta/gamma scale gives narrower confidence intervals. Maybe it's because of the parameterization?

```
set.seed(101)
plot(sirfit, method="wmvrnorm")
plot(sirfit2, add=TRUE, col.traj="red", col.conf="red", method="wmvrnorm")
```