Basic ODE fitting

July 4, 2018

Contents

1 Preliminaries **1**

2 Basic fitting **1**
2.1 Exponential decay model 1
2.2 Chemical reaction - multiple state fitting 5

1 Preliminaries

Load packages:

```
library(fitode)
library(dplyr)
library(tidyr)

## Warning:  package 'tidyr' was built under R version 3.3.3

library(ggplot2); theme_set(theme_bw())

## Warning:  package 'ggplot2' was built under R version 3.3.2

library(rbenchmark)
```

2 Basic fitting

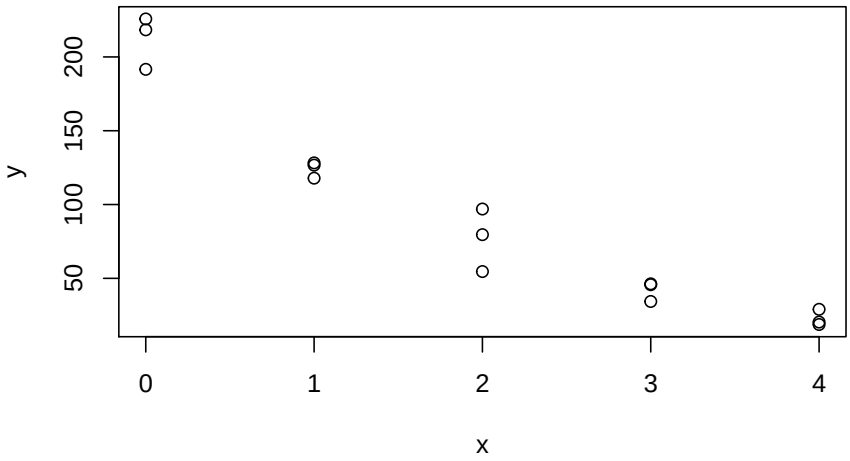
2.1 Exponential decay model

Suppose we have a quantity that is decreasing exponentially.

```
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")
)
```

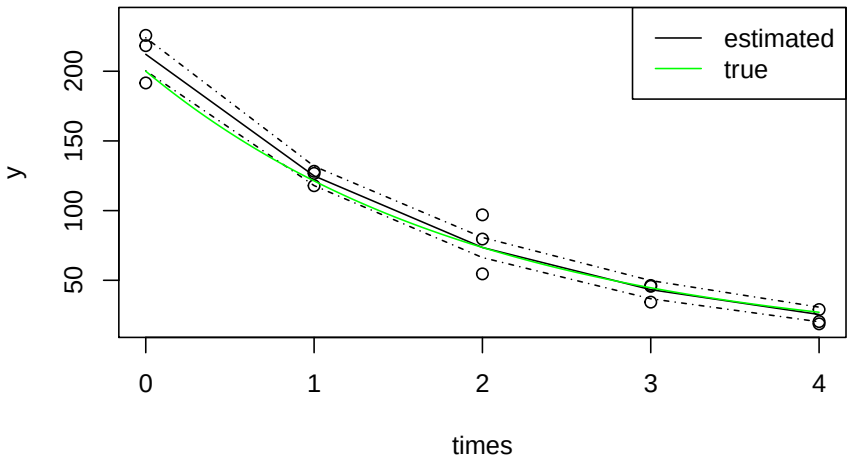
Then, we can fit the model:

```
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 `plot` function. Using the `level` argument will plot 95% confidence intervals of the true trajectory, estimated via delta method.

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



To obtain the confidence interval, we can use `confint` function. 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   0.4727404   0.5880111
## A0 212.1596895 200.3075885 224.0117905
## sd  11.0526632   7.0975903  15.0077361
```

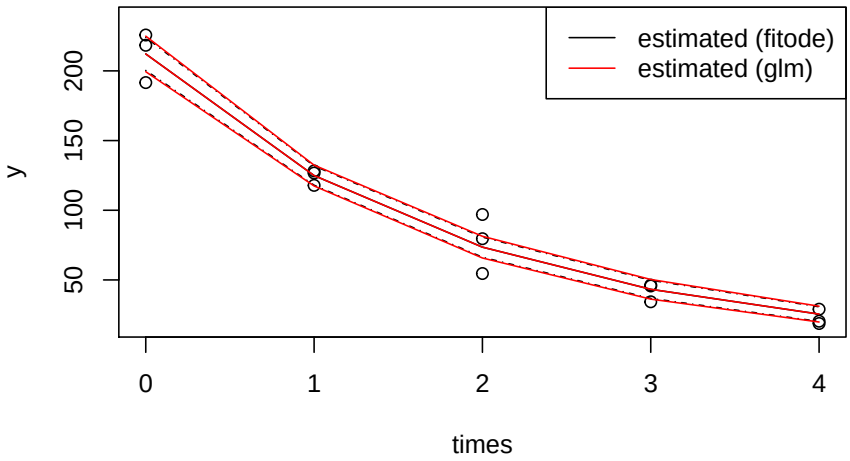
Note that in this particular example, we can use `glm` function to fit the model as well. We can compare the results.

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

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
)
```



Predicted trajectories and their confidence intervals are essentially indistinguishable.

2.2 Chemical reaction - multiple state fitting

Now, consider the following chemical reaction:

$$A \to 3B$$

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

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

Suppose we have measured both quantities and have data:

```
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` meausres quantity A and `y2` measures quantity B. Then, we can define the model:

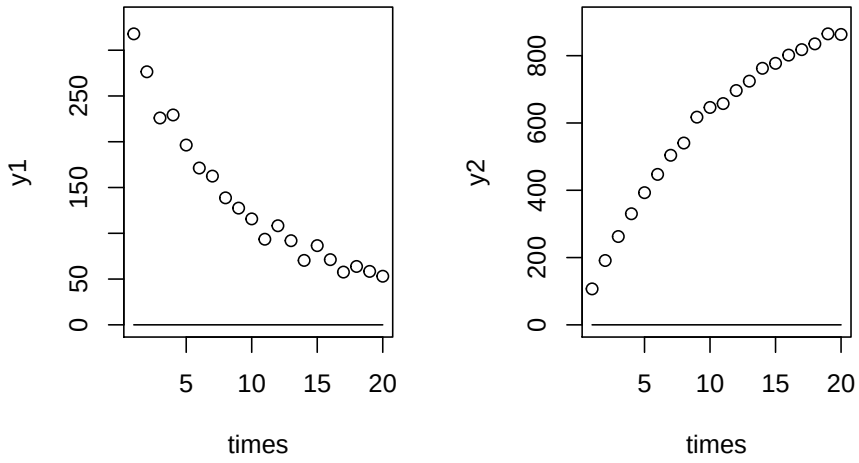
```
reaction_model <- new("model.ode",
    name = "reaction",
    model = list(
        A ~ - k * A,
        B ~ 3 * k * A
    ),
    observation = list(
        y1 ~ dnorm(mean=A, sd=sd1),
        y2 ~ dnorm(mean=B, sd=sd2)
    ),
    initial = list(
        A~A0,
        B~B0
    ),
    par=c("k", "A0", "B0", "sd1", "sd2")
)
```

We can fit this using arbitrary starting conditions.

```
reaction_fit <- fitode(
    reaction_model,
    reaction_data,
    start=c(k=0.2, A0=300, B0=10, sd1=1, sd2=1)
)

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

plot(reaction_fit)
```



Note that optimizer may go to wrong places if we start with bad starting conditions. In that case, we can try starting at different values. Here's how you can set up a simple Latin Hypercube Sampling algorithm.

```
ranges <- data.frame(
    k=c(0.01, 0.2),
    A0=c(300, 350),
    B0=c(90, 110),
    sd1=c(10, 20),
    sd2=c(10,20)
)

startpar <- apply(ranges, 2, function(x) seq(from=x[1], to=x[2], length.out=5))
startpar[] <- apply(startpar, 2,sample)

reaction_fitlist <- apply(startpar, 1, function(x){
    suppressMessages(fitode(
        reaction_model,
        reaction_data,
        start=x
    ))
})
```

We obtain a better fit:

```
reaction_max <- reaction_fitlist[[which.max(sapply(reaction_fitlist, logLik))]]
plot(reaction_max)
```