Basic SIR fitting

June 12, 2017

1 Preliminaries

Load packages:

```
library(fitsir)
library(dplyr)
library(ggplot2); theme_set(theme_bw())
```

2 Harbin

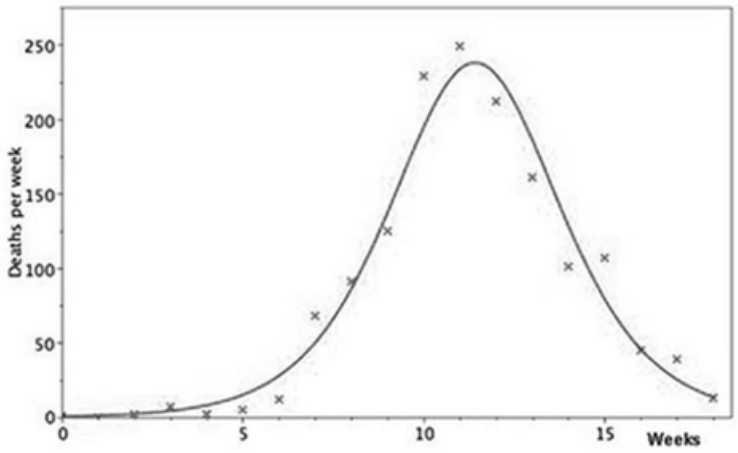


Figure 1: Unnumbered figure (p. 102) from **?** showing the Harbin epidemic.

Figure 1 shows a Kermack-McKendrick model fit to Harbin plague data. Based on the equations (1) below and estimates ("$x_0 = 2985$, $\mathcal{R}_0 = 2.00$ and a mean infectious period of 11 days") that **?** provides, we can compare the

Kermack-McKendrick model to an SIR model fit based on maximum likelihood estimation.

$$\begin{aligned} \frac{dz}{dt} &= \frac{\gamma x_0}{2\mathcal{R}_0^2} c_1 \mathrm{sech}^2(c_1 \gamma t - c_2), \\ c_1 &= \sqrt{(\mathcal{R}_0 - 1)^2 + \frac{2\mathcal{R}_0^2}{x_0}} \\ c_2 &= \tanh^{-1}\left(\frac{\mathcal{R}_0 - 1}{c_1}\right). \end{aligned} \tag{1}$$

[1]

The `harbin` data set is built in:

```
head(harbin)
```

```
##   week Deaths
## 1    2      2
## 2    3      7
## 3    4      2
## 4    5      6
## 5    6     12
## 6    7     68
```

We transform the parameters provided by **?** into *unconstrained parameters* (`log.beta`, `log.gamma`, `log.N`, codelogit.i) so that they can be used as starting parameters for `fitsir`. Although `fitsir` expects a dataframe with column names `times` and `count`, we can specify a time column and a count column with `tcol` and `icol` arguments.

```
dietz_harbin <- c(x0=2985,rzero=2,gamma=7/11)
dietz_lpars <- with(as.list(dietz_harbin),
      c(log.beta=log(rzero*gamma),
        log.gamma=log(gamma),
        log.N=log(x0),
        logit.i=qlogis(1e-3)))
(ff <- fitsir(harbin, start=dietz_lpars, type="death",
              tcol="week", icol="Deaths", method="BFGS"))
```

```
##
## Call:
## mle2(minuslogl = objfun, start = start, method = method, data = dataarg,
##     vecpar = TRUE, gr = gradfun, control = control)
##
```

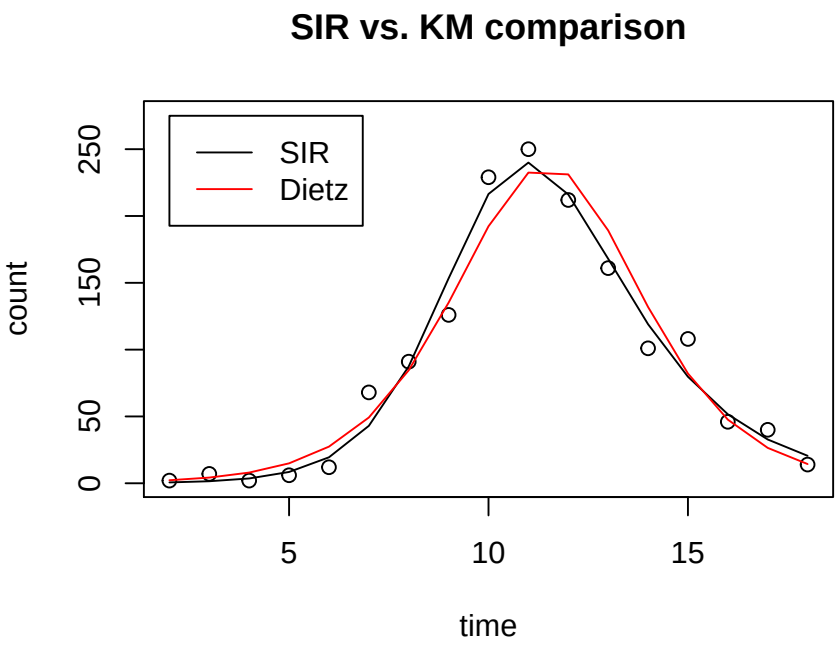
[1] The original equation provided by **?** contains a typo. $c_1\gamma t$ after sech^2 in the first equation should be corrected to $c_1\gamma t/2$ (**?**).

```
## Coefficients:
##   log.beta  log.gamma      log.N    logit.i
##  0.4868478 -0.2708639  7.4966582 -8.1274230
##
## Log-likelihood: -68.27
```

In this case, BFGS method has been used because using sensitivity equations allows for more accurate computation of the Hessian matrix as well as faster convergence.

We can plot `fitsir` objects using `plot` function to see whether this fit is good or not (`plot(ff)`). Here, we plot the SIR fit along with the Dietz fit to compare:

```
plot(ff, main="SIR vs. KM comparison")
times <- harbin$week
dpKM <- with(as.list(dietz_harbin),
        {
           c1 <- sqrt((rzero-1)^2+2*rzero^2/x0)
           c2 <- atanh((rzero-1)/c1)
           gamma*x0/(2*rzero^2)*c1*
               (1/cosh(c1*gamma*times/2-c2))^2
        })
lines(times,dpKM, col = 2)
legend(x=2, y=275, legend=c("SIR","Dietz"), col=c("black", "red"), lty = 1)
```



Apart from the difference in the estimated trajectories, we note that the Kermack-Mckendrick equation models the instantaneous change in the number of recovered individuals (dR/dt) whereas `fitsir` fits are based on the actual number of individuals that recovered during a given time interval ($R(\tau_{n+1}) - R(\tau_n)$).

We can also use the `summary` method provided by the `fitsir` package to see the summarized parameters:

```
summary(ff)

## Maximum likelihood estimation
##
## Call:
## mle2(minuslogl = objfun, start = start, method = method, data = dataarg,
##     vecpar = TRUE, gr = gradfun, control = control)
##
## Coefficients:
##                    R0          r     infper         i0         I0
## Estimate   2.1334e+00 8.6446e-01 1.3111e+00 2.9524e-04 5.3203e-01
## Std. Error 5.4710e-01 1.0344e-01 4.9281e-01 1.2155e-04 2.6501e-01
##                    S0       N
## Estimate   1.8015e+03 1802.01
## Std. Error 2.6259e+02  262.77
##
```

```
## -2 log L: 136.5431
```

MLE returns slightly higher $\mathcal{R}_0$ and longer infectious period but lower population size.

2.1 Overdispersion

In fact, this is not the best fit. By looking at the sum of Pearson residuals divided by the mean (given by `dispersion`), we can see that the data is over dispersed

```
dispersion(ff)

## [1] 4.056574
```

`fitsir` provides three ways of dealing with overdispersion (quasipossion, NB1, NB2) and in this case, using NB1 error function fits better (higher Log-likelihood) than using any of the provided error functions.

First, to explore how these fits differ, we define a new data frame, namely `harbin2`, to avoid using `tcol` and `icol` arguments:

```
harbin2 <- setNames(harbin, c("times", "count"))
```

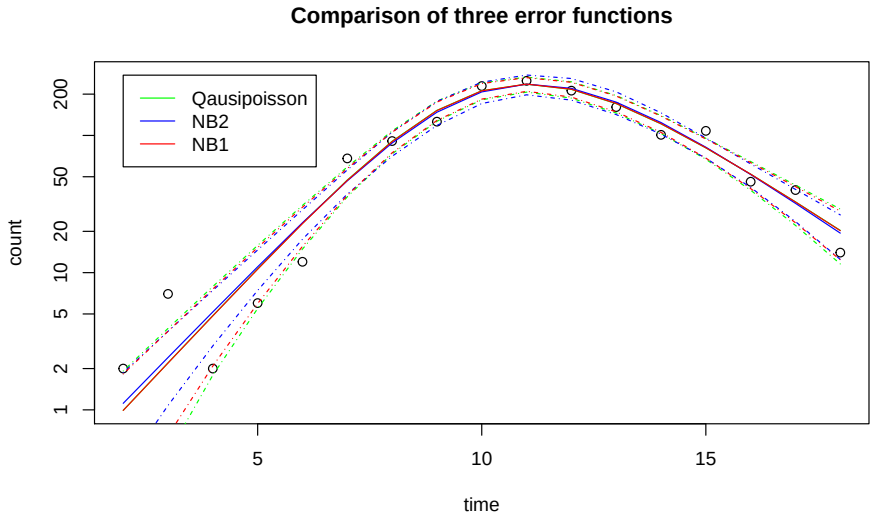
Then, we can fit:

```
ff2 <- fitsir(harbin2, dist="quasipoisson", type="death", method="BFGS", start=dietz_lpars)
ff3 <- fitsir(harbin2, dist="nbinom", type="death", method="BFGS", start=c(dietz_lpars, ll.k
ff4 <- fitsir(harbin2, dist="nbinom1", type="death", method="BFGS", start=c(dietz_lpars, ll.
```

Notice that `nbinom` and `nbinom1` estimate 5 parameters rather than 4. The last parameters are the log-transformed dispersion parameters. NB2 (`nbinom`) assumes quadratic mean-variance relationship: $\text{Var} = \mu(1 + \mu/k)$. In this case, we estimate `ll.k=exp(k)` instead. On the other hand, NB1 (`nbinom1`) assumes linear mean-variance relationship: $\text{Var} = \mu(1 + \phi)$. In this case, we estimate `ll.phi=exp(phi)` instead.

Again, we can plot these three fits to compare:

```
plot(ff2, level=0.95, col.traj="green", col.conf="green", log="y", main="Comparison of three
plot(ff3, level=0.95, add=TRUE, col.traj="blue", col.conf="blue")
plot(ff4, level=0.95, add=TRUE, col.traj="red", col.conf="red")
legend(x=2, y=275, legend=c("Qausipoisson","NB2", "NB1"), col=c("green", "blue", "red"), lty
```

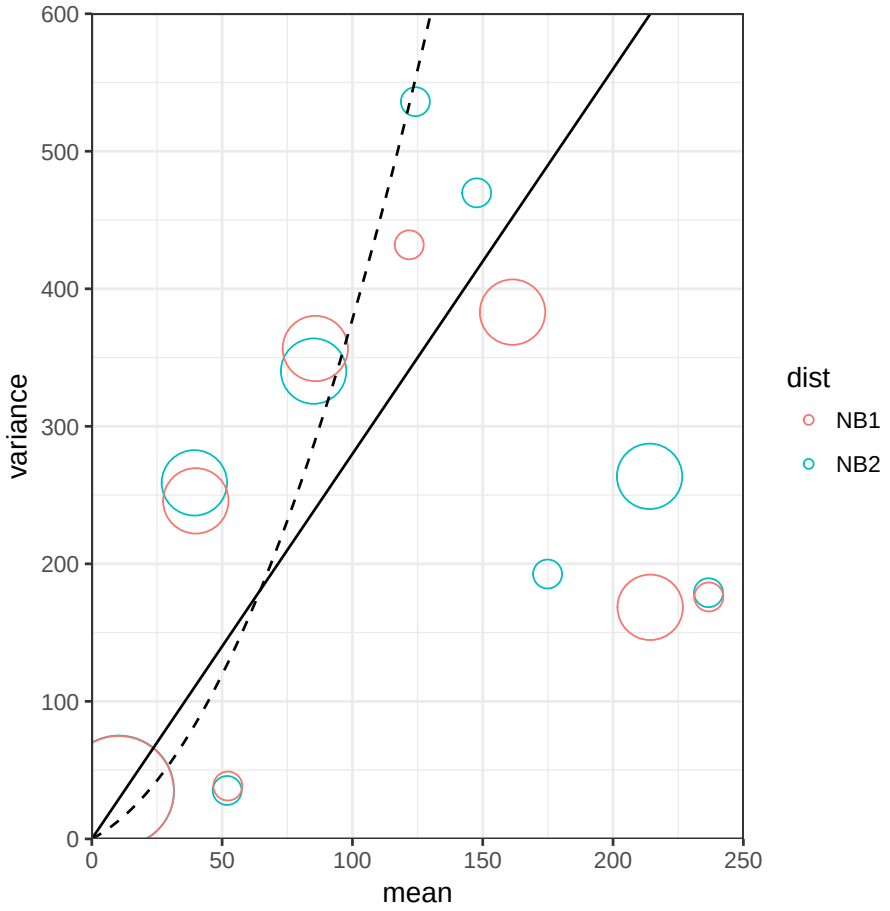


All these three fits give us very similar expected trajectories as well as confidence intervals. However, if we compare their AIC valuess, we find that NB1 gives us the best fit.

```
hfits <- list(QP=ff2, NB2=ff3, NB1=ff4)
lapply(hfits, AIC)
```

```
## $QP
## [1] 151.4747
##
## $NB2
## [1] 145.3402
##
## $NB1
## [1] 139.2647
```

To understand why NB1 fits better than NB2, we can look at the mean variance relationship (we disregard quasipoisson due to its high log likelihood value).



Clearly, we can see that the quadratic mean-variance relationship is not appropriate in this case.

Summarizing the best fit, we underestimate $\mathcal{R}_0$ as well as the population size but the estimate of the infectious period is very close to that provided by Dietz.

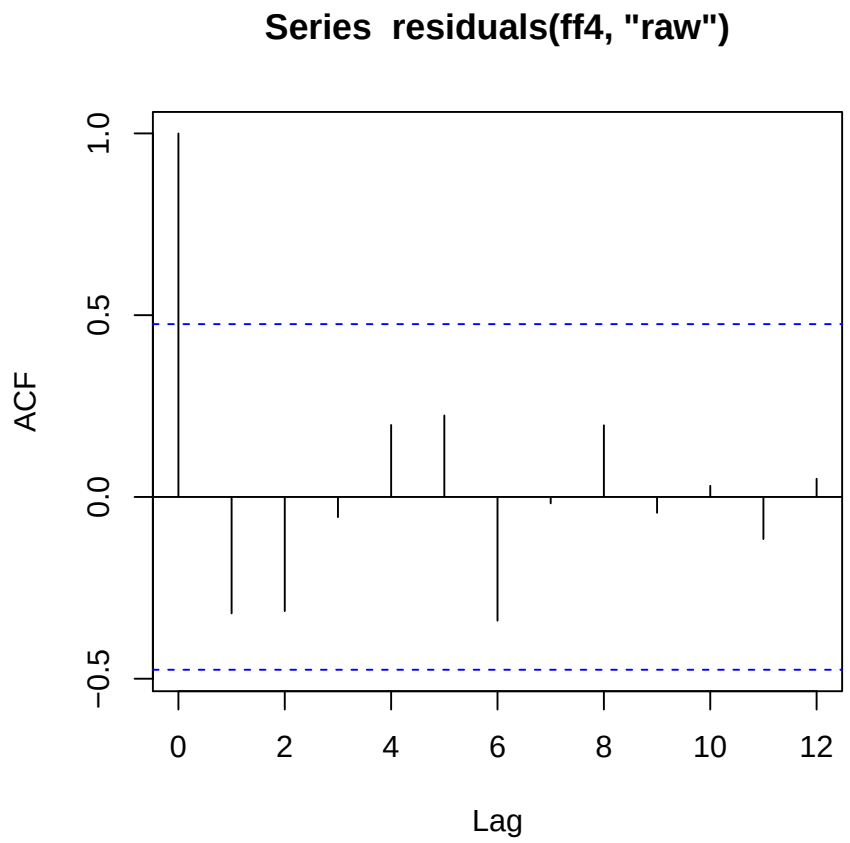
```
summary(ff4)

## Maximum likelihood estimation
##
## Call:
## mle2(minuslogl = objfun, start = start, method = method, data = dataarg,
##     vecpar = TRUE, gr = gradfun, control = control)
##
## Coefficients:
##                    R0          r     infper         i0         I0
## Estimate   1.8630e+00 7.9799e-01 1.0815e+00 3.5120e-04 7.0063e-01
```

```
## Std. Error 4.1009e-01 7.3268e-02 4.2881e-01 1.3290e-04 2.2956e-01
##                     S0       N
## Estimate   1.9943e+03 1994.97
## Std. Error 3.6751e+02  367.51
##
## -2 log L: 129.2647
```

2.2 Autocorrelation

All three models (quasipoisson, NB1, and NB2) provide extremely similar trajectories as well as confidence intervals. So we can test for auto correlation:

```
acf(residuals(ff4, "raw"))
```



This doesn't seem like they're autocorrelated...

3 1918 Philadelphia Flu

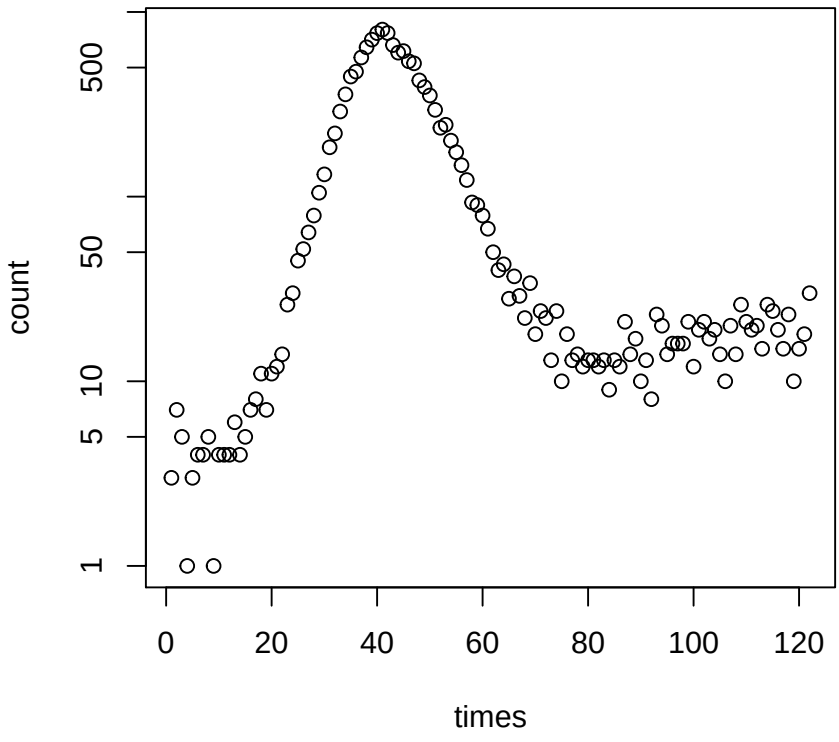
Another data set provided by the `fitsir` package is 1918 philadelphia flu data.

```
head(phila1918)
```

```
##         date pim
## 1 1918-09-01   3
## 2 1918-09-02   7
## 3 1918-09-03   5
## 4 1918-09-04   1
## 5 1918-09-05   3
## 6 1918-09-06   4
```

Notice that the first column is in the `Date` format. Since `fitsir` expects a time column to be a numeric vector, we can either add a new column or create a new data frame. Here, we create a new data frame to avoid using `tcol` and `icol` argument.

```
phila1918a <- with(phila1918, data.frame(times=seq_along(date), count=pim))
plot(phila1918a, log="y")
```

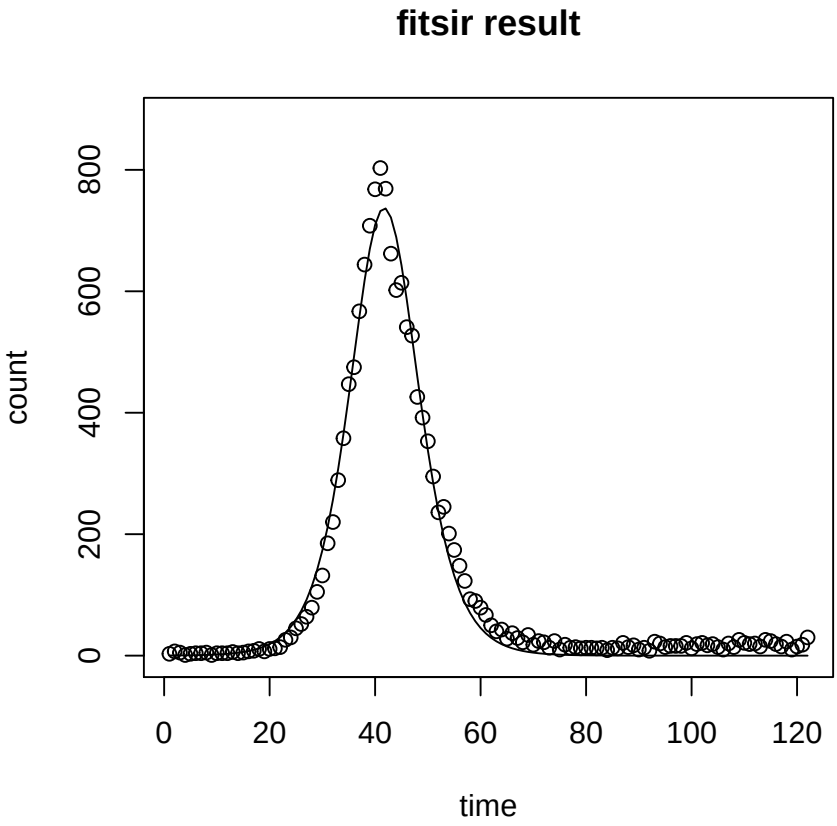


Notice that this data doesn't exactly follow the typical SIR trajectory. Due to its long tail, most models other than Gaussian fail.

For the Harbin data, we used Dietz parameter as a starting parameter but we can't do that here. First, we can try to fit an SIR model using arbitrary startin values:

```
(pfit <- fitsir(phila1918a,
    start=c(log.beta=log(0.12), log.gamma=log(0.09), log.N=log(10000), logit.i=qlogis(0.01)),
    type="death")
)

##
## Call:
## mle2(minuslogl = objfun, start = start, method = method, data = dataarg,
##     vecpar = TRUE, gr = gradfun, control = control)
##
## Coefficients:
```

```
##   log.beta  log.gamma      log.N    logit.i
##   1.609535   1.561885  11.854347 -14.996334
##
## Log-likelihood: -559.23

plot(pfit)
```

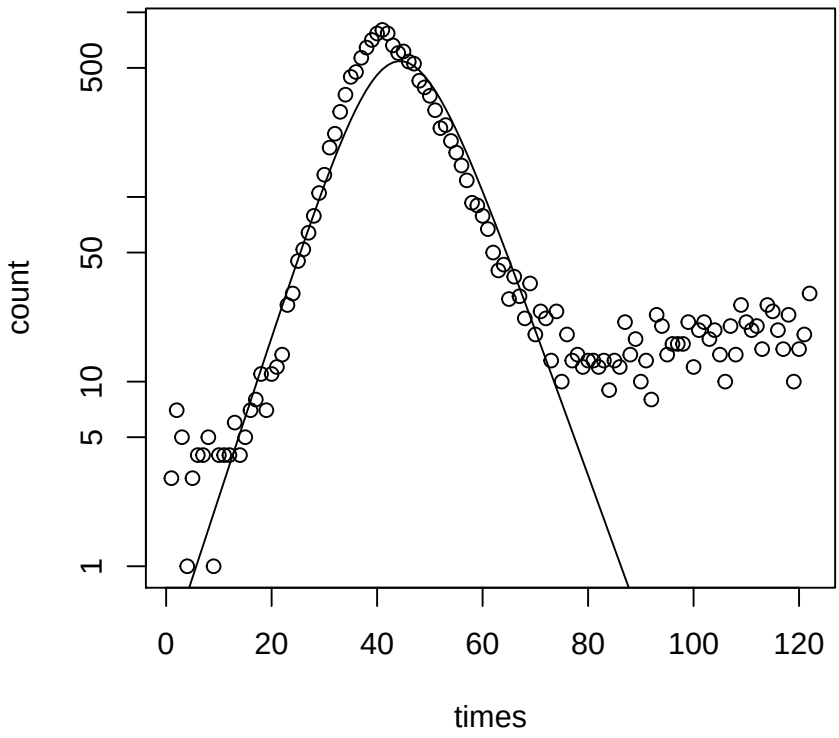


We get a decent fit. However, this is actually not the maximum likelihood estimate. To avoid falling in a local minima, we can either use a different starting point or try different method. Here, we present how using different starting values can yield different results.

3.1 Starting values

`fitsir` provides a function (`startfun`) that automatically finds a reasonable starting value:

```
(pstart <- startfun(phila1918a, type="death"))

##    log.beta   log.gamma       log.N     logit.i
##   0.4056933   0.2621874  10.7081521 -12.0159633

plot(phila1918a, log="y")
lines(SIR.detsim(phila1918a$times, trans.pars(pstart), type="death"))
```



Using this starting function, we can get a better fit:

```
(pfit2 <- fitsir(phila1918a, type="death", start=pstart))

##
## Call:
## mle2(minuslogl = objfun, start = start, method = method, data = dataarg,
##     vecpar = TRUE, gr = gradfun, control = control)
##
```

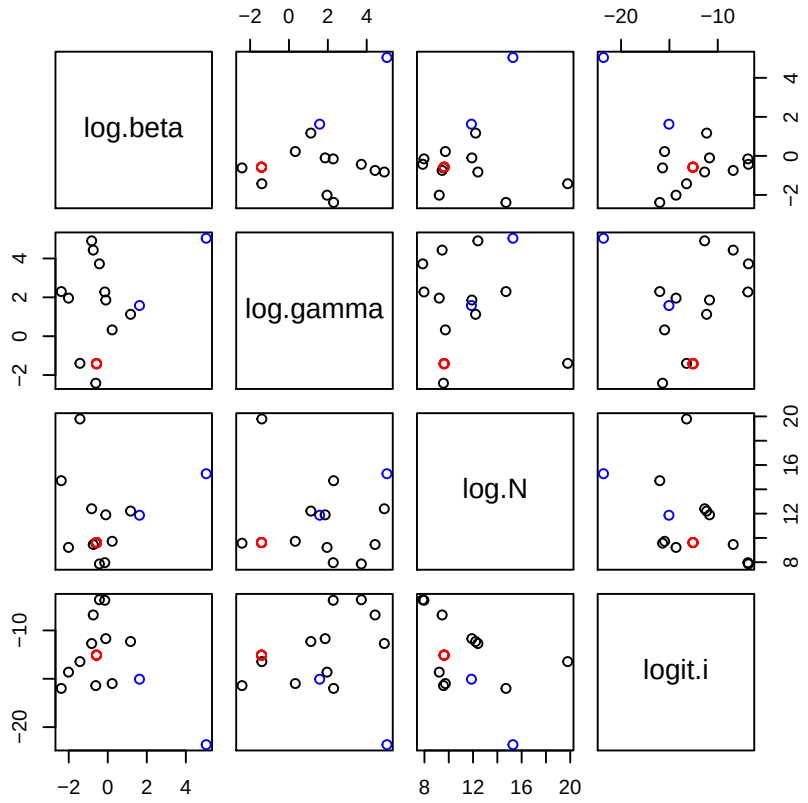
```
## Coefficients:
##    log.beta   log.gamma       log.N     logit.i
##   0.4194201   0.2440950  10.6548521 -12.8742423
##
## Log-likelihood: -549.81
```

Yet, this is still not the best fit. We can further explore different starting values using Latin hypercube sampling.

```
set.seed(123)
lhsfun <- function(param, size=0.5, length.out=20) {
    seq((1-size)*param, (1+size)*param, length.out=length.out)
}

ltab <- sapply(pstart, lhsfun)
ltab <- apply(ltab,2, sample)
plist <- apply(ltab, 1, function(x) fitsir(phila1918a, type="death", start=x))
(pLik <- sapply(plist, logLik))

##  [1] -512.1789 -836.0653 -836.0653 -512.1786 -836.0654 -816.2729 -622.1658
##  [8] -836.0653 -562.4931 -836.0654 -512.1786 -559.2596 -512.1787 -836.0653
## [15] -512.1786 -512.1786 -705.8693 -836.0653 -512.1786 -836.0654
```

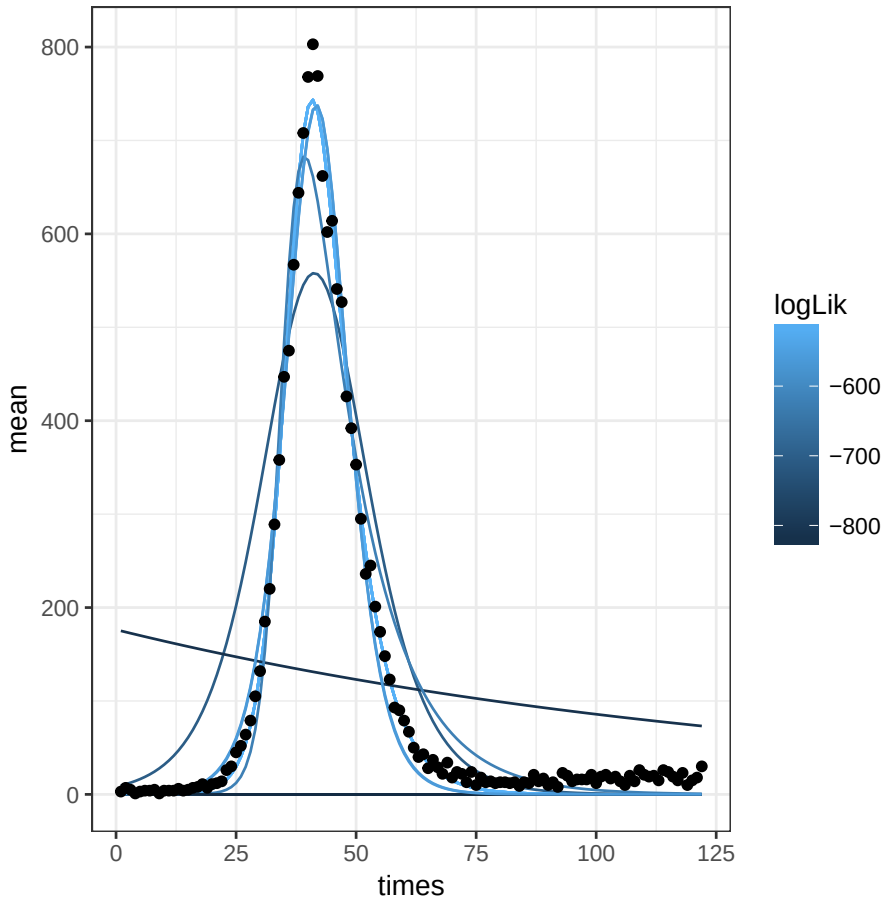
We can explore the parameter space

```
ppars <- as.data.frame(t(sapply(plist, coef)))
col <- c("black", "blue", "red")
ccol <- col[cut(pLik, breaks=c(-900, -600, -520, -500))]

plot(ppars, col=ccol)
```



We can see that the best fits (red points) converge to a single point. We get some fits that are not optimal but are close to the best fits (blue points). Then, there are fits that fail (black points).

```
ppred <- plist %>%
    lapply(predict) %>%
    bind_rows(.id="sim")

ppred$logLik <- rep(pLik, each=length(phila1918a$times))

ggplot(ppred) +
    geom_line(aes(times, mean, group=sim, col=logLik)) +
    geom_point(data=phila1918a, aes(times, count))
```



Looking at the best fit...

```
pbest <- plist[[which.max(pLik)]]
summary(pbest) ## Nelder-Mead being unstable

## Maximum likelihood estimation
##
## Call:
## mle2(minuslogl = objfun, start = start, method = method, data = dataarg,
##     vecpar = TRUE, gr = gradfun, control = control)
##
## Coefficients:
##                    R0          r     infper         i0         I0
## Estimate   2.3023e+00 3.1681e-01 4.1106e+00 3.5175e-06 5.2914e-02
## Std. Error         NA 5.1283e-04         NA 3.7657e-08 9.6437e-05
##                    S0       N
## Estimate   1.5043e+04 15042.9
```

```
## Std. Error 1.0010e+02   100.1
##
## -2 log L: 1024.357

pbest2 <- fitsir(phila1918a, type="death", start=coef(pbest), method="BFGS")
summary(pbest2)

## Maximum likelihood estimation
##
## Call:
## mle2(minuslogl = objfun, start = start, method = method, data = dataarg,
##     vecpar = TRUE, gr = gradfun, control = control)
##
## Coefficients:
##                    R0          r     infper         i0         I0
## Estimate   2.3023e+00 3.1681e-01 4.1106e+00 3.5175e-06 5.2914e-02
## Std. Error 1.3162e-01 8.6558e-03 3.1455e-01 7.4611e-07 1.2462e-02
##                    S0        N
## Estimate   1.5043e+04 15042.87
## Std. Error 4.0122e+02   401.23
##
## -2 log L: 1024.357
```

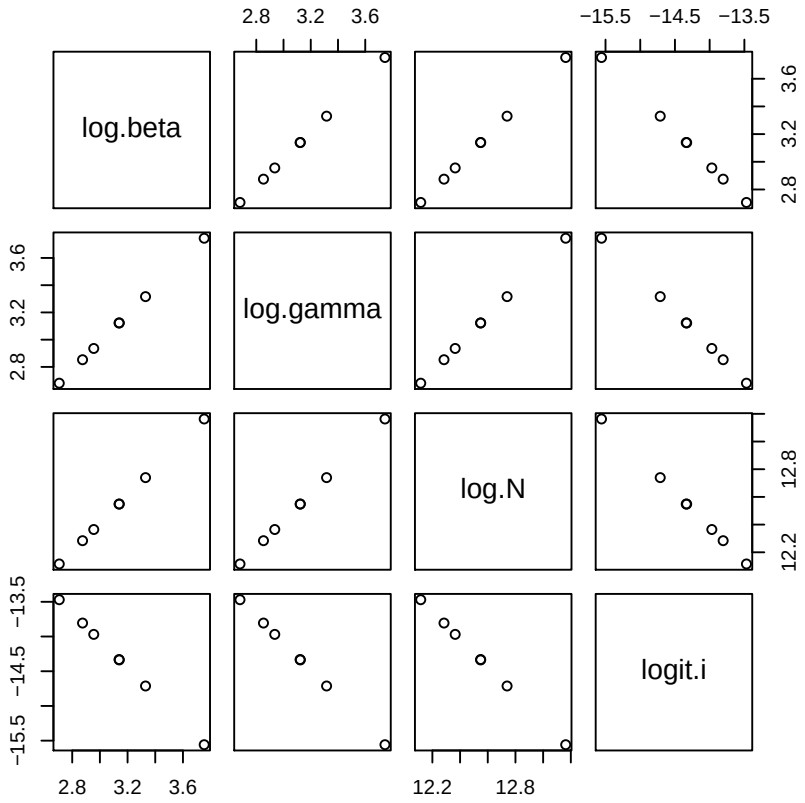
4 Bombay

In this section, we are going to be demonstrating some pathologies that might be associated with fitting an SIR model.

```
bombay2 <- setNames(bombay, c("times", "count"))
bbstart <- startfun(bombay2, type="death")
bb <- fitsir(bombay2, type="death", dist="nbinom", start=c(bbstart, ll.k=5))
```

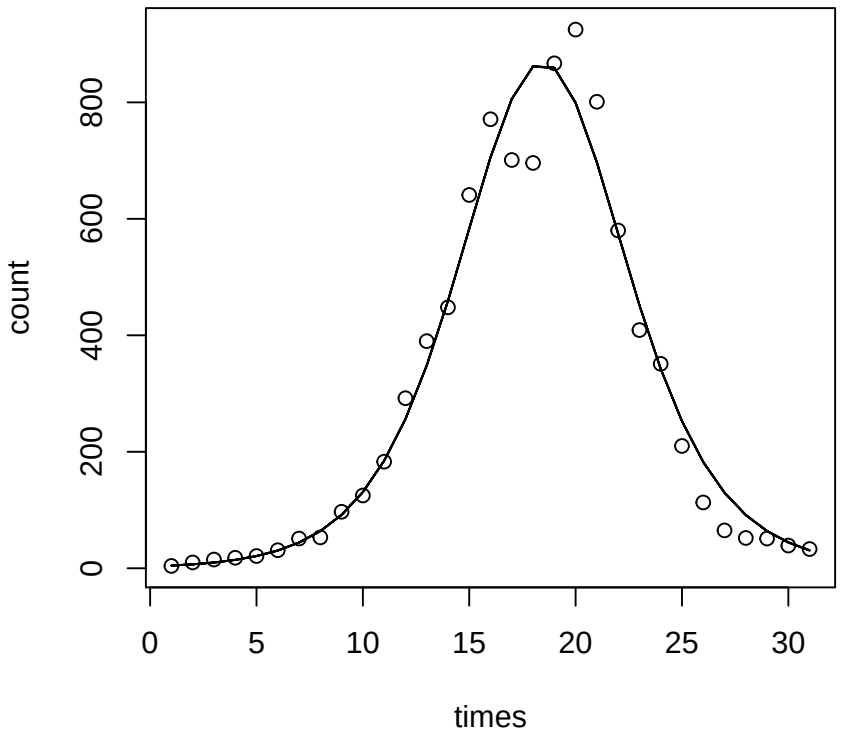
Multiple local minima:

```
ltab <- sapply(bbstart, lhsfun)
ltab <- apply(ltab,2, sample)
blist <- apply(ltab, 1, fitsir,
               data=bombay2, type="death", method="BFGS")
bpars <- as.data.frame(t(sapply(blist, coef)))
bLik <- sapply(blist, logLik)
mle <- max(bLik)
goodfits <- which(1.001 * mle < bLik)
gpars <- bpars[goodfits,]
plot(gpars)
```



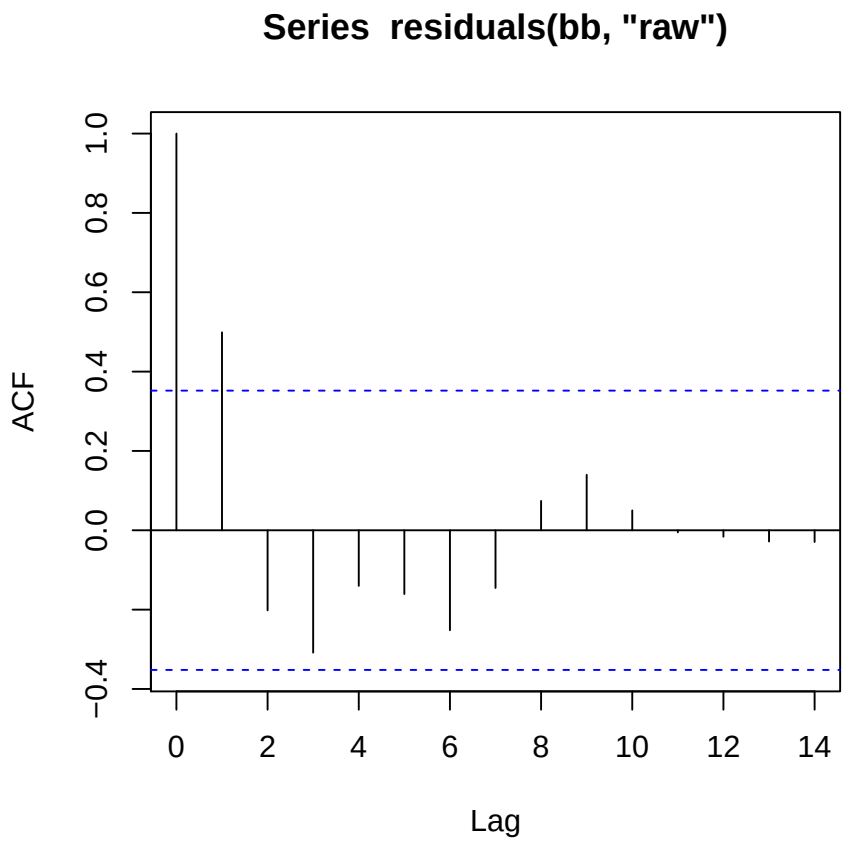
The difference is not noticeable:

```
plot(bombay2)
l <- lapply(blist[goodfits], function(x) plot(x, add=TRUE))
```



Autocorrelation:

```
acf(residuals(bb, "raw"))
```


Series residuals(bb, "raw")
ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.4
0
2
4
6
8
10
12
14
Lag

References