An introduction to *outbreaker* 1.1-0

Thibaut Jombart

December 13, 2013

Abstract

This vignette introduces the main functionalities of *outbreaker* [1], a package implementing a model for disease outbreak reconstruction using epidemiological data and pathogen genome sequences. The emphasis of this document is put on using *outbreaker* and on the visualization and interpretation of results.

More resources for disease outbreaks analysis in R are available on the *R-epi project*: http://sites.google.com/site/therepiproject. Questions about *outbreaker*, alonside any question relating to the analysis of epidemics, should be asked on the R-epi forum. To do so, send an email to: r-epi@googlegroups.com, or visit the website: http://groups.google.com/forum/#!forum/r-epi.

Contents

1 Running outbreaker

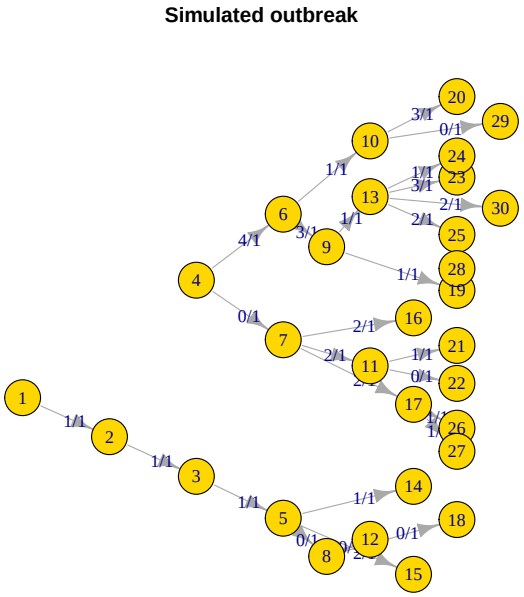
1.1 A simple example

In this vignette, we shall use the toy dataset `fakeOutbreak` distributed with *outbreaker*. We first load the package, the dataset, and look at the object's content:

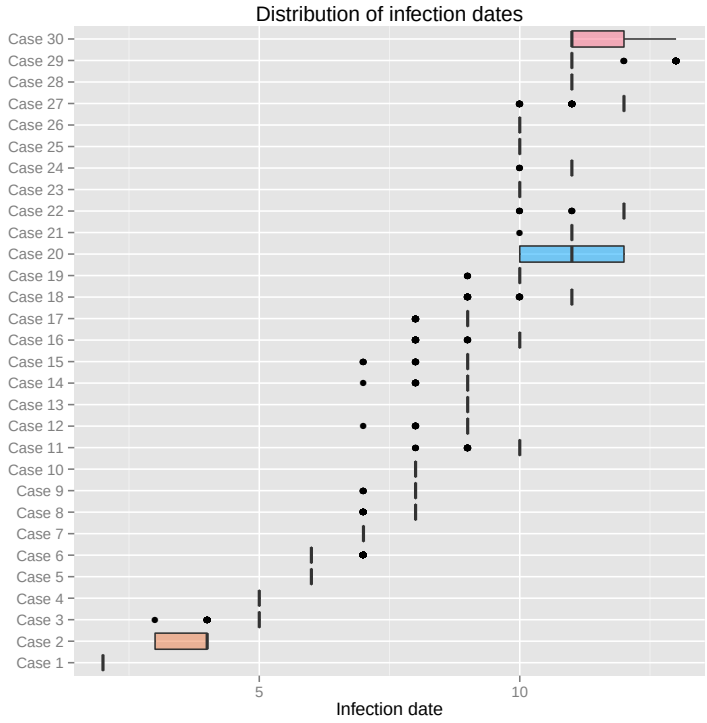
```
library(outbreaker)
data(fakeOutbreak)
class(fakeOutbreak)

## [1] "list"

names(fakeOutbreak)

## [1] "dat"         "w"           "collecDates" "res"

class(fakeOutbreak$dat)

## [1] "simOutbreak"

fakeOutbreak$w

## [1] 0.00 0.50 1.00 0.75

fakeOutbreak$collecDates

##  [1]  3  5  6  6  7  9  8  9  9  9 11 10 10 10 10 11 11 12 11 13 12 13 11 12 11
## [26] 11 13 12 14 14
```

`fakeOutbreak` is a list containing `dat`, a simulated outbreak (obtained by `simOutbreak`), a generation time distribution (`w`), collection dates for the DNA sequences (`collecDates`), and results from *outbreaker* (`res`). We shall need only the first 3 items, and re-create the fourth.

```
dat <- fakeOutbreak$dat
w <- fakeOutbreak$w
collecDates <- fakeOutbreak$collecDates
plot(dat, main="Simulated outbreak")
```




```
barplot(w, main="Generation time distribution", ylab="probability", xlab="days", names=0:3)
```



We run *outbreaker* on these data, using the parallel version, and specifying that we want to run 4 MCMC in parallel and 100,000 chains for each MCMC.

```
res <-  outbreaker.parallel(n.runs=4, dna=dat$dna,
                            dates=collecDates,w.dens=w, n.iter=1e5)
```

```
names(res)
```

```
##  [1] "chains"         "collec.dates"   "w"              "f"
##  [5] "D"              "idx.dna"        "tune.end"       "find.import"
##  [9] "burnin"         "find.import.at" "n.runs"         "call"
```

```
names(res$chains)
```

```
##   [1] "step"     "post"     "like"     "prior"    "mu1"      "mu2"
##   [7] "gamma"    "pi"       "spa1"     "spa2"     "Tinf_1"   "Tinf_2"
##  [13] "Tinf_3"   "Tinf_4"   "Tinf_5"   "Tinf_6"   "Tinf_7"   "Tinf_8"
##  [19] "Tinf_9"   "Tinf_10"  "Tinf_11"  "Tinf_12"  "Tinf_13"  "Tinf_14"
##  [25] "Tinf_15"  "Tinf_16"  "Tinf_17"  "Tinf_18"  "Tinf_19"  "Tinf_20"
##  [31] "Tinf_21"  "Tinf_22"  "Tinf_23"  "Tinf_24"  "Tinf_25"  "Tinf_26"
##  [37] "Tinf_27"  "Tinf_28"  "Tinf_29"  "Tinf_30"  "alpha_1"  "alpha_2"
##  [43] "alpha_3"  "alpha_4"  "alpha_5"  "alpha_6"  "alpha_7"  "alpha_8"
##  [49] "alpha_9"  "alpha_10" "alpha_11" "alpha_12" "alpha_13" "alpha_14"
##  [55] "alpha_15" "alpha_16" "alpha_17" "alpha_18" "alpha_19" "alpha_20"
##  [61] "alpha_21" "alpha_22" "alpha_23" "alpha_24" "alpha_25" "alpha_26"
##  [67] "alpha_27" "alpha_28" "alpha_29" "alpha_30" "kappa_1"  "kappa_2"
##  [73] "kappa_3"  "kappa_4"  "kappa_5"  "kappa_6"  "kappa_7"  "kappa_8"
##  [79] "kappa_9"  "kappa_10" "kappa_11" "kappa_12" "kappa_13" "kappa_14"
##  [85] "kappa_15" "kappa_16" "kappa_17" "kappa_18" "kappa_19" "kappa_20"
##  [91] "kappa_21" "kappa_22" "kappa_23" "kappa_24" "kappa_25" "kappa_26"
##  [97] "kappa_27" "kappa_28" "kappa_29" "kappa_30" "run"
```

```
class(res)
```

```
## [1] "list"
```

```
names(res)
```

```
##  [1] "chains"         "collec.dates"   "w"              "f"
##  [5] "D"              "idx.dna"        "tune.end"       "find.import"
##  [9] "burnin"         "find.import.at" "n.runs"         "call"
```

The object `res` is a list with a number of named items, described in `?outbreaker`. The most important one is `res$chains`, containing the MCMC outputs:

```
class(res$chains)
```

```
## [1] "data.frame"
```

```
dim(res$chains)
```

```
## [1] 804 101
```

```
names(res$chains)
```

```
##   [1] "step"     "post"     "like"     "prior"    "mu1"      "mu2"
##   [7] "gamma"    "pi"       "spa1"     "spa2"     "Tinf_1"   "Tinf_2"
##  [13] "Tinf_3"   "Tinf_4"   "Tinf_5"   "Tinf_6"   "Tinf_7"   "Tinf_8"
##  [19] "Tinf_9"   "Tinf_10"  "Tinf_11"  "Tinf_12"  "Tinf_13"  "Tinf_14"
##  [25] "Tinf_15"  "Tinf_16"  "Tinf_17"  "Tinf_18"  "Tinf_19"  "Tinf_20"
##  [31] "Tinf_21"  "Tinf_22"  "Tinf_23"  "Tinf_24"  "Tinf_25"  "Tinf_26"
##  [37] "Tinf_27"  "Tinf_28"  "Tinf_29"  "Tinf_30"  "alpha_1"  "alpha_2"
##  [43] "alpha_3"  "alpha_4"  "alpha_5"  "alpha_6"  "alpha_7"  "alpha_8"
##  [49] "alpha_9"  "alpha_10" "alpha_11" "alpha_12" "alpha_13" "alpha_14"
##  [55] "alpha_15" "alpha_16" "alpha_17" "alpha_18" "alpha_19" "alpha_20"
##  [61] "alpha_21" "alpha_22" "alpha_23" "alpha_24" "alpha_25" "alpha_26"
##  [67] "alpha_27" "alpha_28" "alpha_29" "alpha_30" "kappa_1"  "kappa_2"
##  [73] "kappa_3"  "kappa_4"  "kappa_5"  "kappa_6"  "kappa_7"  "kappa_8"
##  [79] "kappa_9"  "kappa_10" "kappa_11" "kappa_12" "kappa_13" "kappa_14"
##  [85] "kappa_15" "kappa_16" "kappa_17" "kappa_18" "kappa_19" "kappa_20"
##  [91] "kappa_21" "kappa_22" "kappa_23" "kappa_24" "kappa_25" "kappa_26"
##  [97] "kappa_27" "kappa_28" "kappa_29" "kappa_30" "run"

res$chains[1:10,1:10]

##    step    post    like prior       mu1       mu2 gamma     pi spa1 spa2
## 1     1 -1106.3 -1108.4 2.093 5.000e-05 5.000e-05     1 0.9770    0    0
## 2   500  -468.0  -470.2 2.144 5.021e-05 5.021e-05     1 0.9826    0    0
## 3  1000  -446.9  -448.8 1.909 5.038e-05 5.038e-05     1 0.9572    0    0
## 4  1500  -446.7  -448.9 2.294 5.060e-05 5.060e-05     1 0.9990    0    0
## 5  2000  -446.5  -448.7 2.227 5.087e-05 5.087e-05     1 0.9916    0    0
## 6  2500  -446.6  -448.7 2.147 5.080e-05 5.080e-05     1 0.9829    0    0
## 7  3000  -446.9  -449.1 2.216 5.138e-05 5.138e-05     1 0.9905    0    0
## 8  3500  -448.5  -450.6 2.168 5.078e-05 5.078e-05     1 0.9852    0    0
## 9  4000  -446.8  -449.1 2.233 5.309e-05 5.309e-05     1 0.9923    0    0
## 10 4500  -452.0  -453.8 1.855 5.323e-05 5.323e-05     1 0.9515    0    0
```

The columns of this `data.frame` store the following outputs:

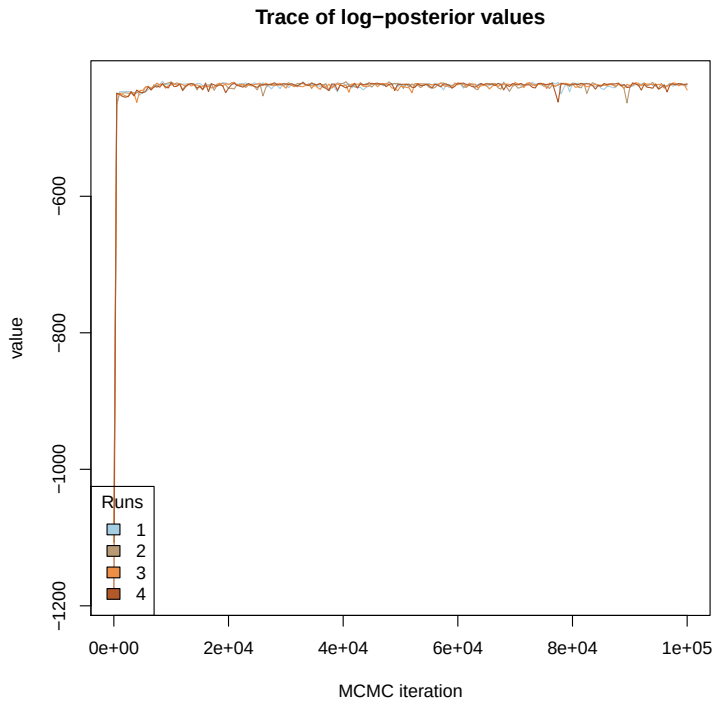
- `step`: the MCMC iteration of the sample
- `post/like/prior`: log values for posterior, likelihood, and prior densities
- `mu1`: in mutation model 1, mutation rate; otherwise, the rate of transitions, per site and generation
- `mu2`: in mutation model 1, mutation rate (mu1=mu2); otherwise, the rate of transversions, per site and generation
- `gamma`: the ratio between transversions and transitions (μ_2/μ_1)
- `pi`: the proportion of the transmission tree sampled
- `Tinf_[number]`: dates of infection
- `alpha_[number]`: the index of the ancestral cases (infectors)
- `kappa_[number]`: the number of generations between cases and their most recent sampled ancestor (here, fixed to 1)
- `run`: for parallel runs, the index of the run.

We shall see how this information can be used, visualized and interpreted over the following sections.

1.2 Assessing convergence and determining the burnin

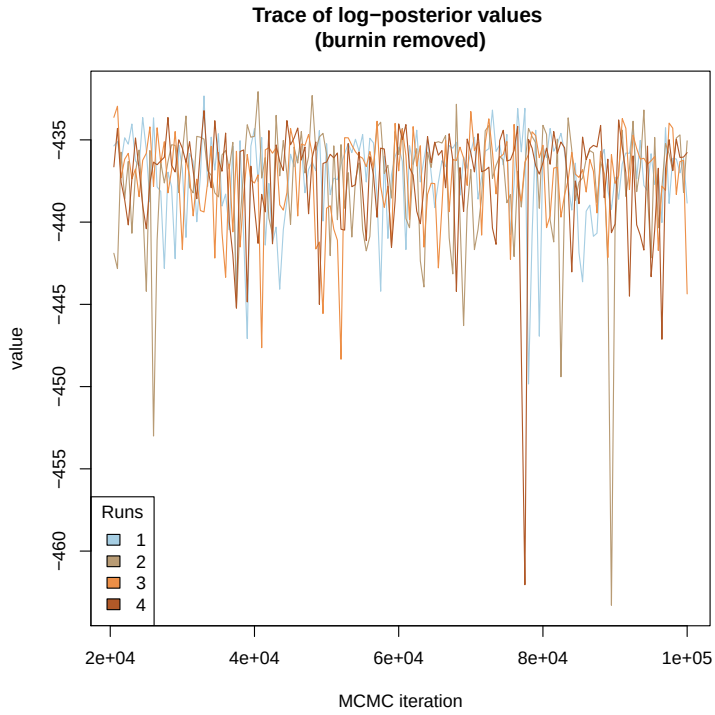
A MCMC is said to converge when it reaches a stationary state, i.e. its distributional properties are constant over time (mean and variance don't depend on which window of the MCMC you consider). Convergence of the chains is best assessed by comparing parallel runs. This can be done using `plotChains`, which we to visualize the trace of the log-posterior values of the model:

```
plotChains(res, main="Trace of log-posterior values")
```

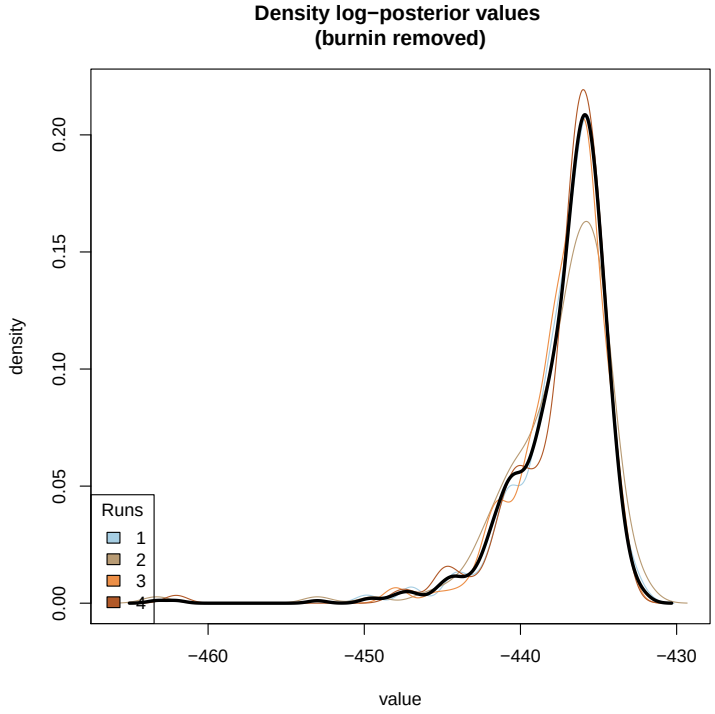


We set the burnin to a conservative 20,000 steps, and plot the MCMC output as trace and as densities:

```
plotChains(res, main="Trace of log-posterior values \n(burnin removed)",
           burnin=2e4)
```




```
plotChains(res, main="Density log-posterior values \n(burnin removed)",
           burnin=2e4, type="dens")
```



Here, all four runs have sampled from the same distribution, confirming convergence of the MCMCs. Note that this can also be tested using a simple ANOVA, which we use to check that log-posterior values do no differ from one run to another:

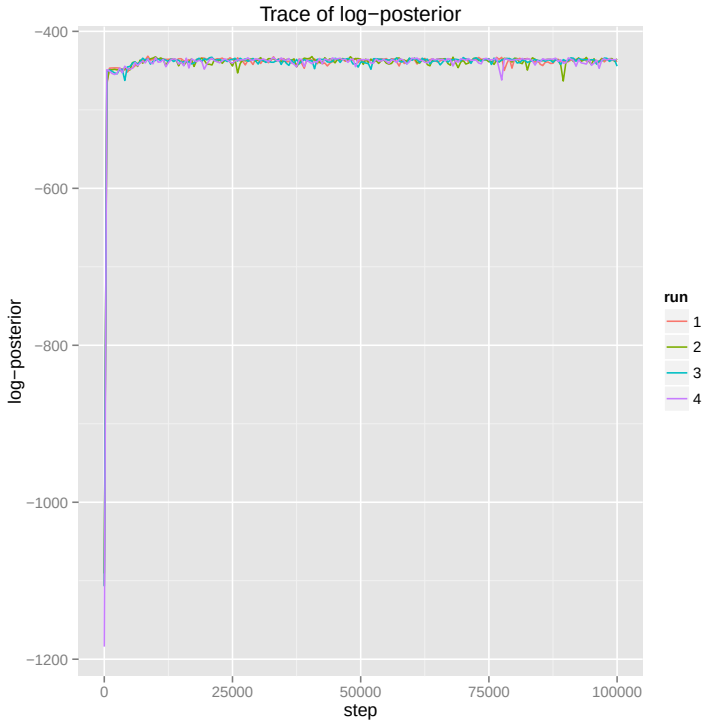
```
anova(lm(post~run, data=res$chains[res$chains$step>2e4,]))

## Analysis of Variance Table
##
## Response: post
##            Df Sum Sq Mean Sq F value Pr(>F)
## run         1      0     0.3    0.03   0.86
## Residuals 638   6387    10.0
```

These graphs can be slightly improved by using *ggplot2* [2]. We first load the package, and make sure that `run` is treated as a `factor`:

```
library(ggplot2)
library(reshape2)
library(mgcv)
x <- res$chains
x$run <- factor(x$run)
```

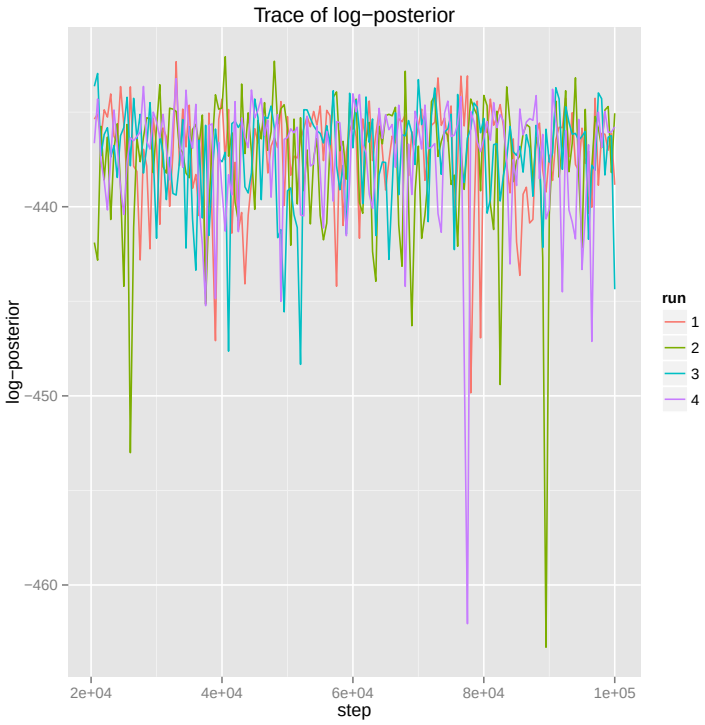
The basic plot of the log-posterior trace is obtained by:

```
p <- ggplot(x, aes(x=step)) +
    geom_line(aes(y=post, colour=run)) +
    labs(title="Trace of log-posterior", y="log-posterior")
p
```



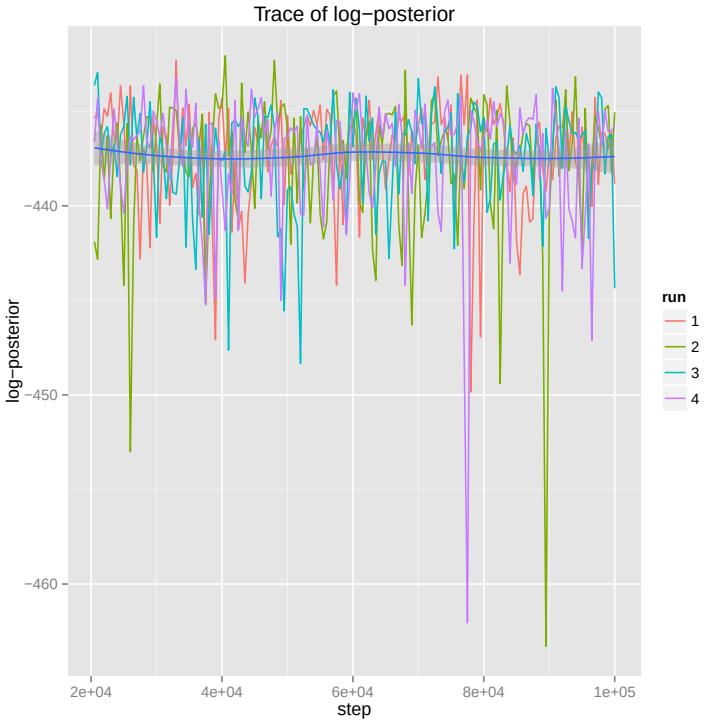
The version without the burnin is:

```
p <- ggplot(x[x$step>2e4,], aes(x=step)) +
    geom_line(aes(y=post, colour=run)) +
    labs(title="Trace of log-posterior", y="log-posterior")
p
```



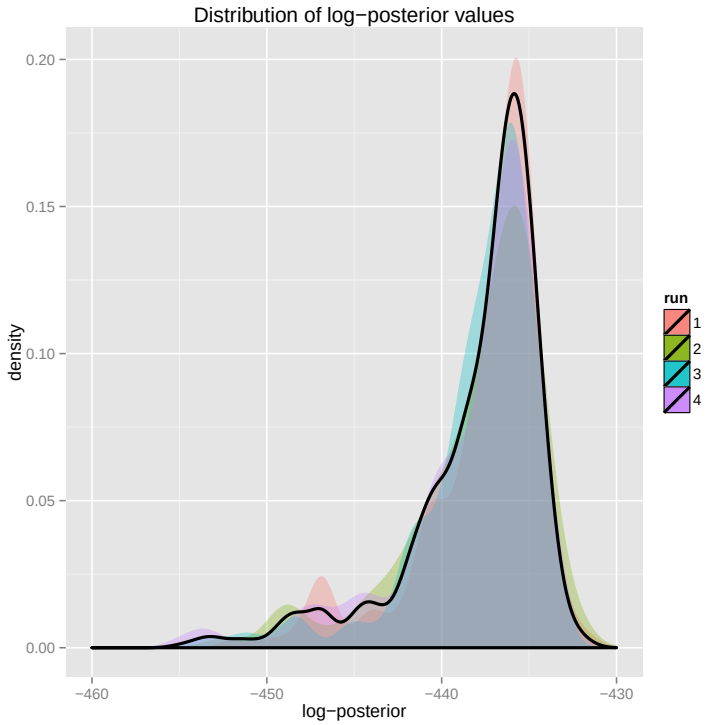
A model of the mean can be added easily:

```
p + geom_smooth(aes(y=post))

## geom_smooth:  method="auto" and size of largest group is <1000, so using loess.  Use
'method = x' to change the smoothing method.
```



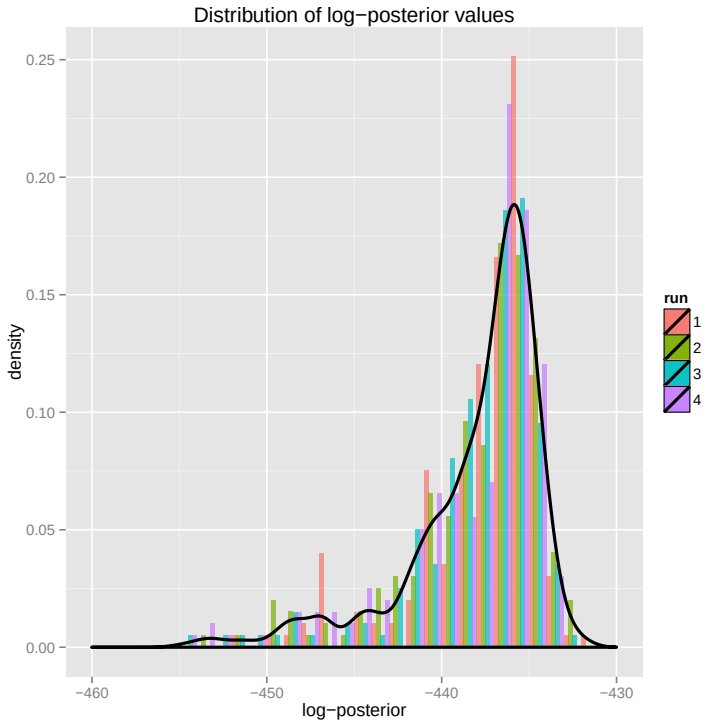
ggplot2 is also pretty good for plotting distributions. Here is an example using 1-dimensional density estimation:

```
p <- ggplot(data=x) + labs(title="Distribution of log-posterior values", x="log-posterior") +
    scale_x_continuous(limits=c(-460,-430))
p + geom_density(aes(x=post, fill=run), alpha=.3, colour=NA) +
    geom_density(aes(x=post), size=1, colour="black", shape=2, alpha=.8)
```



and another version using histograms:

```
p + geom_histogram(aes(x=post, fill=run, y=..density..), alpha=.7, colour=NA, position="dodge") +
    geom_density(aes(x=post), size=1, colour="black", shape=2, alpha=.8)

## stat_bin:  binwidth defaulted to range/30.  Use 'binwidth = x' to adjust this.
```



2 Interpreting the results

outbreaker can provide information on:

- the transmission tree ("who infected whom"); ancestry for case i is noted α_i
- the dates of infection; for case i, noted T_i^{inf}
- the mutation rate per generation of infection (μ)
- the proportion of the outbreak sampled (π)
- effective reproduction numbers over time or at an individual level †
- incidence curves †
- the mutation rate per unit of time †

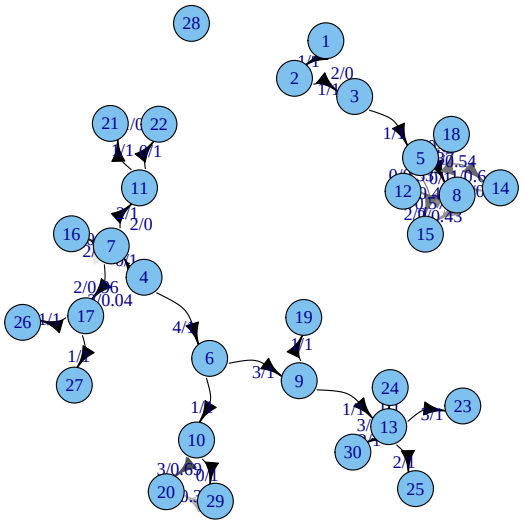
Items indicated with a † are not explicitely modelled by *outbreaker*, but can be derived from the posterior samples of trees and parameters.

2.1 Visualizing reconstructed transmission trees

outbreaker being a Bayesian approach, it does not return a single tree, but a distribution of plausible transmission trees. A set of ancestries can be visualized by `transGraph`

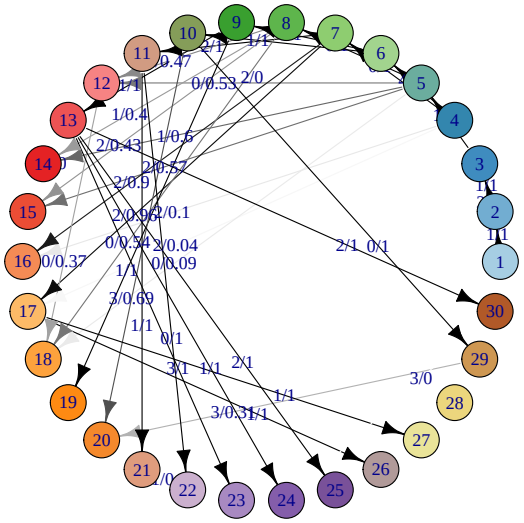
```
library(igraph)
library(adegenet)
```

```
g <- transGraph(res, thres=0)
```



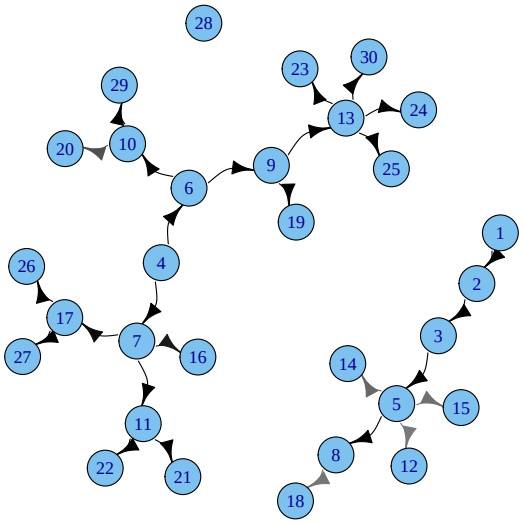
Annotations represent the number of mutations for the ancestries, and their support (frequency in the posterior samples). Note that this function returns an *igraph* object, which can be used for further plotting and customization:

```
g

## IGRAPH DN-- 30 40 --
## + attr: name (v/c), dates (v/n), label (v/n), color (e/c), support
##   (e/n), curved (e/x), nb.mut (e/n), label (e/c)

plot(g, layout=layout.circle, edge.curved=FALSE, vertex.color=funky(30))
```

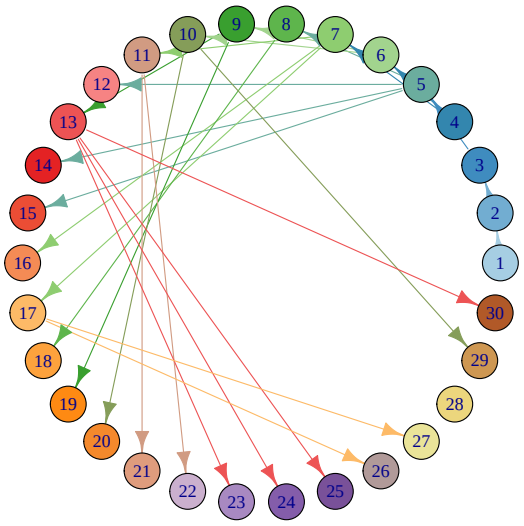


see `?plot.igraph` and `igraph.plotting` for more information on how to customize these graphics. We illustrate some possibilities below:

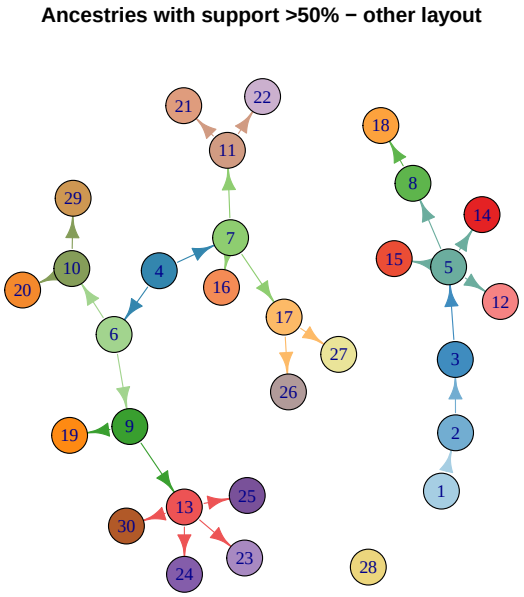
```
g <- transGraph(res, thres=0.5, annot="")
```



```
edge.colors <- funky(30)[as.numeric(get.edgelist(g)[,1])]
plot(g, layout=layout.circle, edge.curved=FALSE, vertex.color=funky(30),
     edge.color=edge.colors)
title("Ancestries with support >50% - circular graph")
```

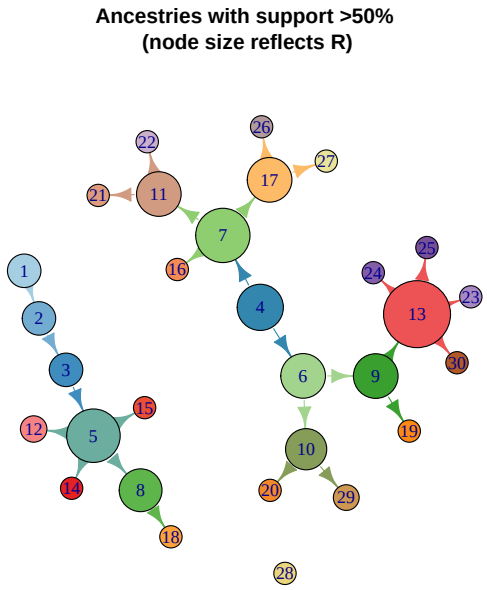



```
plot(g, layout=layout.auto, edge.curved=FALSE, vertex.color=funky(30),
     edge.color=edge.colors)
title("Ancestries with support >50% - other layout")
```



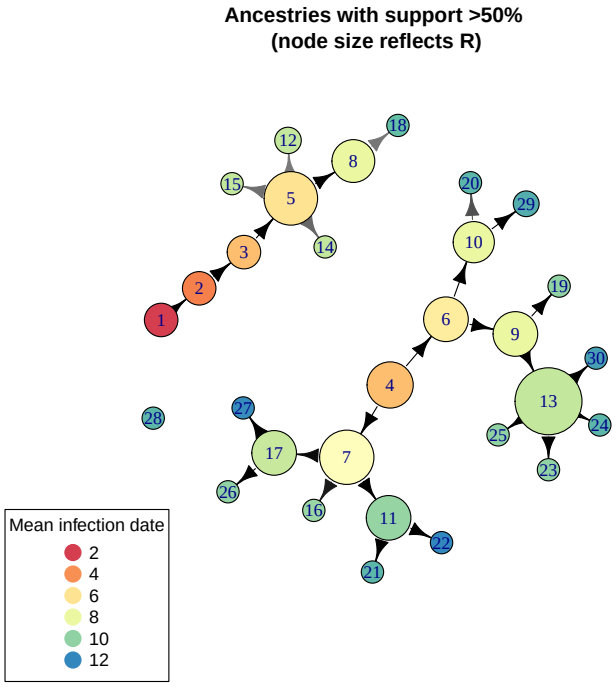
Here we ensure that cases with higher reproduction numbers look bigger:

```
case.size <- 10+apply(get.R(res),2,mean)*5
plot(g, layout=layout.auto, edge.curved=FALSE, vertex.color=funky(30),
     edge.color=edge.colors, vertex.size=case.size)
title("Ancestries with support >50% \n(node size reflects R)")
```



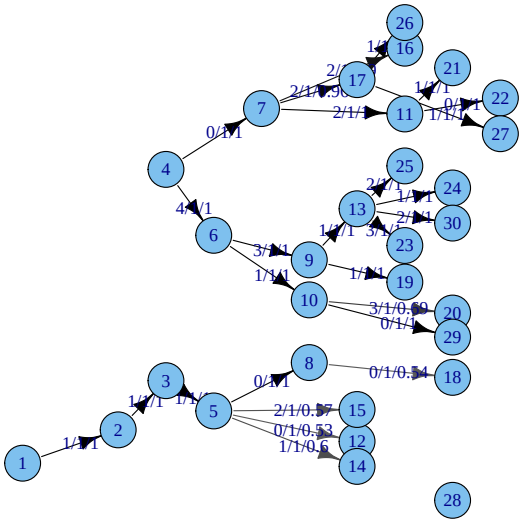
Same idea, but this time colors represent dates of infection:

```
Tinf <- x[x$step>=2e4,grep("Tinf", names(x))]
case.color <- any2col(apply(Tinf,2,mean), col.pal=spectral)
plot(g, layout=layout.auto, edge.curved=FALSE, vertex.color=case.color$col,
     vertex.size=case.size)
title("Ancestries with support >50% \n(node size reflects R)")
legend("bottomleft", col=case.color$leg.col, leg=case.color$leg.txt,
       title="Mean infection date", pch=20, pt.cex=3, inset=-.1)
```



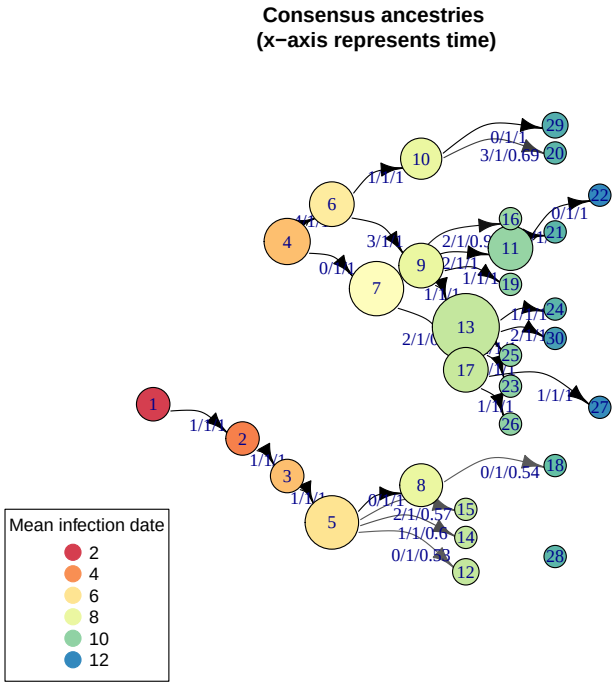
The same can be done with a tree formed by consensus ancestries, obtained by `get.tTree`:

```
plot(get.tTree(res), main="Consensus ancestries - basic plot")
```



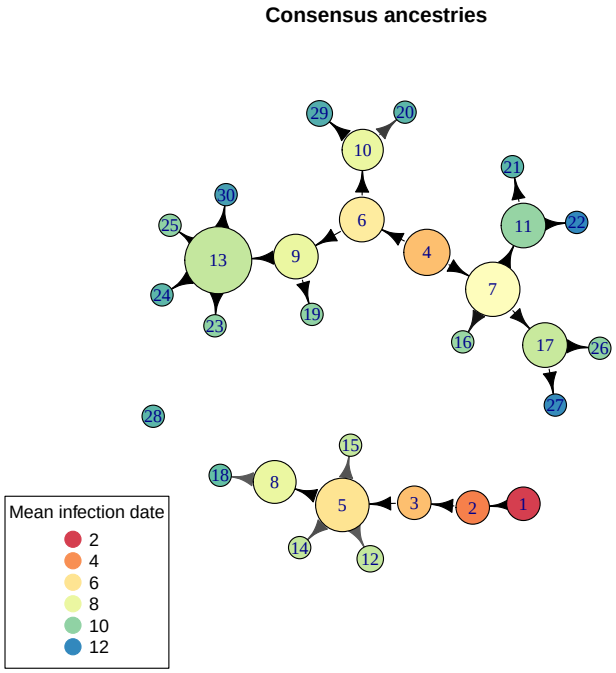
For a customized version where colors represent the mean infection dates:

```
tre <- get.tTree(res)
plot(tre, edge.curved=TRUE, vertex.color=case.color$col,
     vertex.size=case.size)
title("Consensus ancestries \n(x-axis represents time)")
legend("bottomleft", col=case.color$leg.col, leg=case.color$leg.txt,
       title="Mean infection date", pch=20, pt.cex=3, inset=-.1)
```



and another, without the time axis:

```
g <- as.igraph(get.tTree(res))
plot(g, edge.curved=FALSE, vertex.color=case.color$col, layout=layout.auto,
     vertex.size=case.size, edge.label="")
title("Consensus ancestries")
legend("bottomleft", col=case.color$leg.col, leg=case.color$leg.txt,
       title="Mean infection date", pch=20, pt.cex=3, inset=-.1)
```



Note that all these plots can be visualized interactively using `tkplot` (just replace `plot` with `tkplot` in the above command lines). In all these graphs, we can see that cases 1, 4 and 28 have been classified as imported cases. For a more systematic assessment of imported cases, we can just look for cases for which the ancestor is unknown (NA) in the consensus tree:

```
temp <- get.tTree(res)
temp$ances

##  [1] NA  1  2 NA  3  4  4  5  6  6  7  5  9  5  5  7  7  8  9 10 11 11 13 13 13
## [26] 17 17 NA 10 13

which(is.na(temp$ances))

## [1]  1  4 28
```

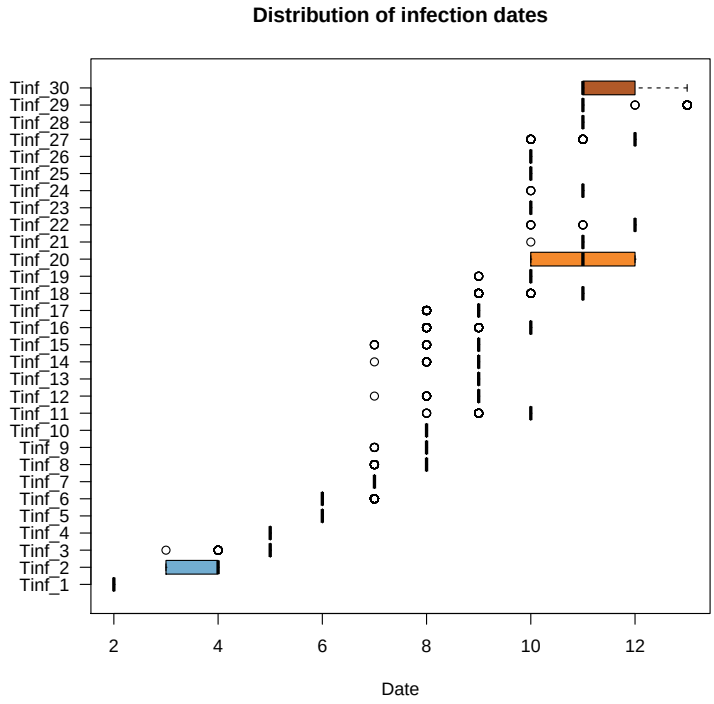
2.2 Plotting dates of infection

Dates of infection are stored in the table of MCMC outputs, with columns starting with `Tinf`:

```
Tinf <- x[x$step>=2e4,c(1,ncol(x),grep("Tinf", names(x)))]
Tinf[1:5,1:6]

##     step run Tinf_1 Tinf_2 Tinf_3 Tinf_4
## 41 20000   1      2      4      5      5
## 42 20500   1      2      4      5      5
## 43 21000   1      2      4      5      5
## 44 21500   1      2      4      5      5
## 45 22000   1      2      3      5      5
```

This information can be visualized easily using the basic `boxplot`:

```
boxplot(Tinf[,-c(1,2)], horizontal=TRUE, las=1, xlab="Date",
        main="Distribution of infection dates", col=funky(30))
```

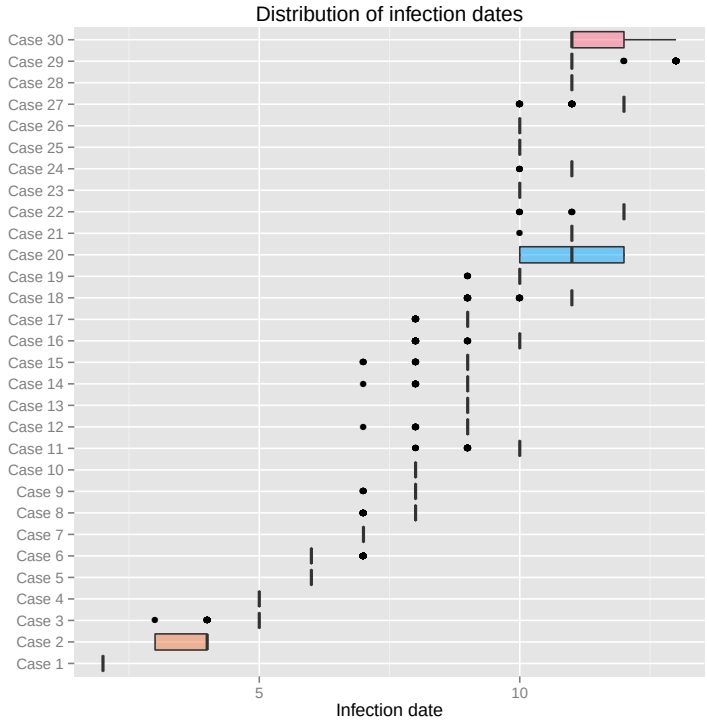


We can also use *ggplot2*, but this demands a slight reformating of the data:

```
Tinf <- melt(Tinf, id=1:2)
names(Tinf)[3:4] <- c("case", "date")
Tinf$case <- sub("Tinf_","Case ", Tinf$case)
Tinf$case <- factor(Tinf$case, levels=paste("Case",1:30))
head(Tinf)

##    step run   case date
## 1 20000   1 Case 1    2
## 2 20500   1 Case 1    2
## 3 21000   1 Case 1    2
## 4 21500   1 Case 1    2
## 5 22000   1 Case 1    2
## 6 22500   1 Case 1    2

tail(Tinf)

##         step run    case date
## 19315  97500   4 Case 30   11
## 19316  98000   4 Case 30   12
## 19317  98500   4 Case 30   12
## 19318  99000   4 Case 30   12
## 19319  99500   4 Case 30   12
## 19320 100000   4 Case 30   11
```

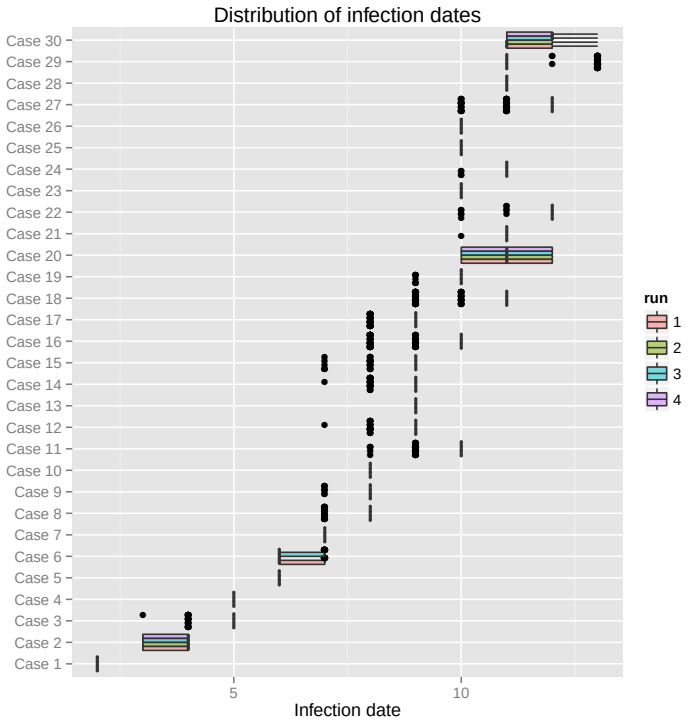
In this case, little is gained in the new plot:

```
p <- ggplot(data=Tinf) + geom_boxplot(aes(x=case,y=date,fill=case), alpha=.5) +
   coord_flip() + labs(y="Infection date", x="", title="Distribution of infection dates")
p + guides(fill=FALSE)
```



However, it would be simple to compare infection dates of different runs, which may be useful if different runs provide slightly different results:

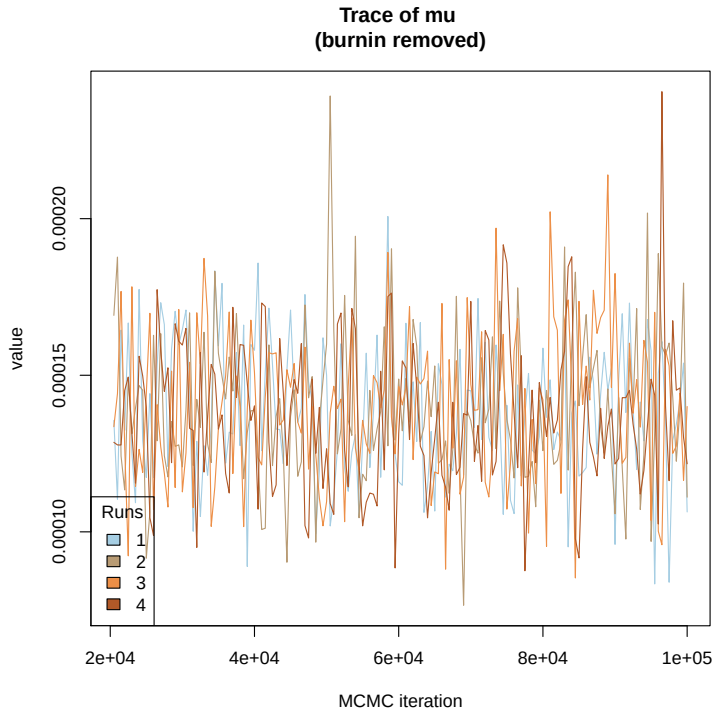
```
ggplot(data=Tinf) + geom_boxplot(aes(x=case,y=date,fill=run),alpha=.5) + coord_flip() +
    labs(y="Infection date", x="", title="Distribution of infection dates")
```



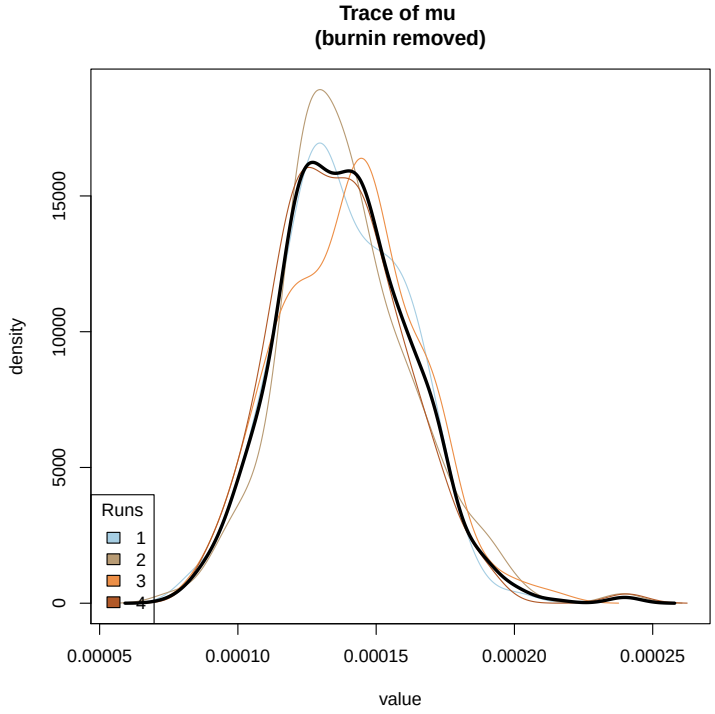
2.3 Accessing posterior distributions

Any element of the model in `res$chains` can be plotted using `plotChains`: it just needs to be named in the argument `what`. For instance, the mutation rate:

```
plotChains(res, main="Trace of mu \n(burnin removed)",
           burnin=2e4, what="mu1")
```




```
plotChains(res, main="Trace of mu \n(burnin removed)",
           burnin=2e4, what="mu1", type="dens")
```



(note that in this case, the plotted information is the mutation rate *per generation of infection*, and not per unit of time. See section on mutation rates below for an estimation of the rates per unit of time.

To derive statistics for a given distribution, one just needs to extract the relevant column, making sure to remove the burnin:

```
mu <- res$chains$mu1[res$chains$step>2e4]
head(mu)

## [1] 0.0001356 0.0001104 0.0001644 0.0001324 0.0001667 0.0001252

length(mu)

## [1] 640

head(mu)

## [1] 0.0001356 0.0001104 0.0001644 0.0001324 0.0001667 0.0001252

summary(mu)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 7.66e-05 1.22e-04 1.38e-04 1.39e-04 1.54e-04 2.41e-04
```

Here we have 640 values to estimate parameters of the distribution of *mu*. However, the effective sample size might be smaller if successive values are correlated (autocorrelated chains). This can be tested easily:

```
cor(mu[-length(mu)], mu[2:length(mu)])

## [1] 0.02841

cor.test(mu[-length(mu)], mu[2:length(mu)])

##
##  Pearson's product-moment correlation
##
## data:  mu[-length(mu)] and mu[2:length(mu)]
## t = 0.7173, df = 637, p-value = 0.4735
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.04926  0.10574
## sample estimates:
##     cor
## 0.02841
```

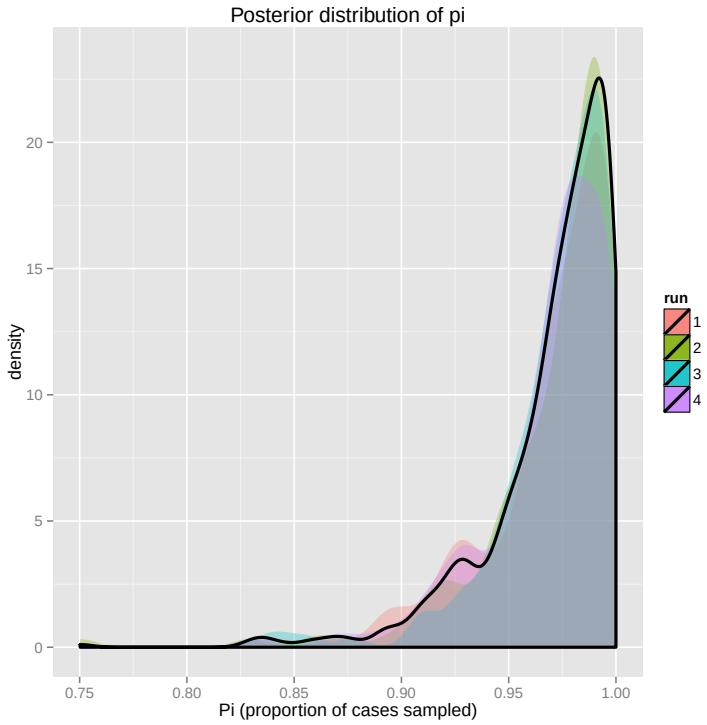
No, there is no correlation between successive values.

As before, `ggplot2` versions of the plot can be obtained; here, for the parameter π (proportion of the outbreak sampled):

```
library(ggplot2)
library(reshape2)
x <- res$chains[x$step>2e4,]
x$run <- factor(x$run)
```

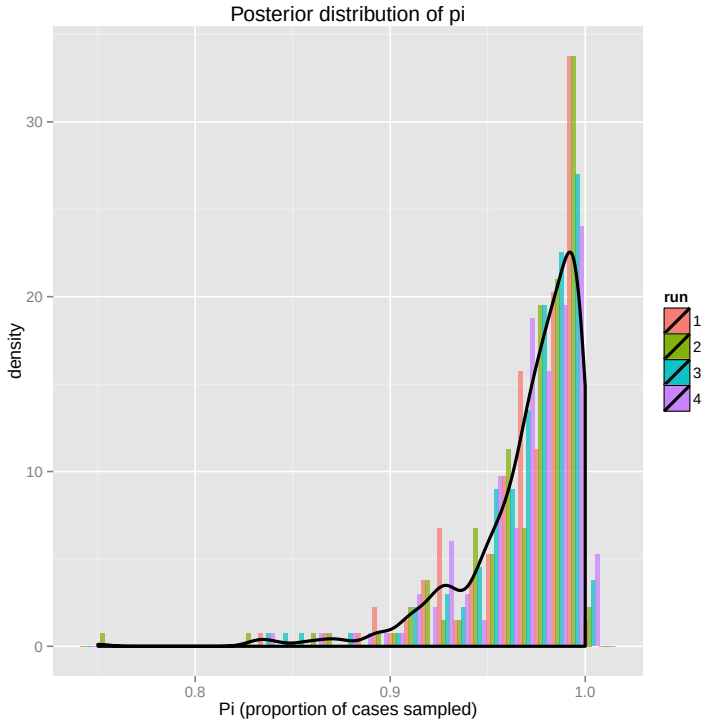
To plot densities:

```
p <- ggplot(data=x) + labs(title="Posterior distribution of pi", x="Pi (proportion of cases sampled)")
p + geom_density(aes(x=pi, fill=run), alpha=.3, colour=NA) +
    geom_density(aes(x=pi), size=1, colour="black", shape=2, alpha=.8)
```



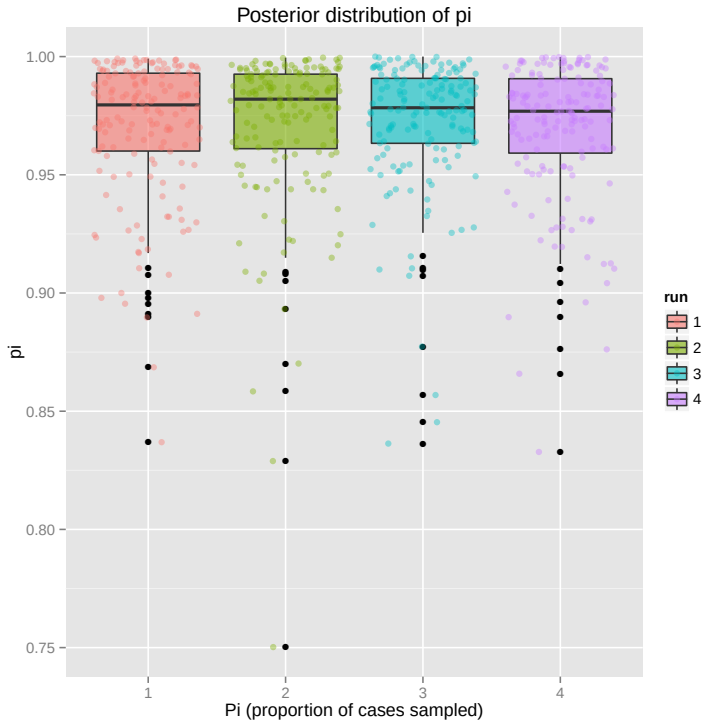
Histograms:

```
p + geom_histogram(aes(x=pi, fill=run, y=..density..), alpha=.7, colour=NA, position="dodge") +
    geom_density(aes(x=pi), size=1, colour="black", shape=2, alpha=.8)

## stat_bin:  binwidth defaulted to range/30.  Use 'binwidth = x' to adjust this.
```



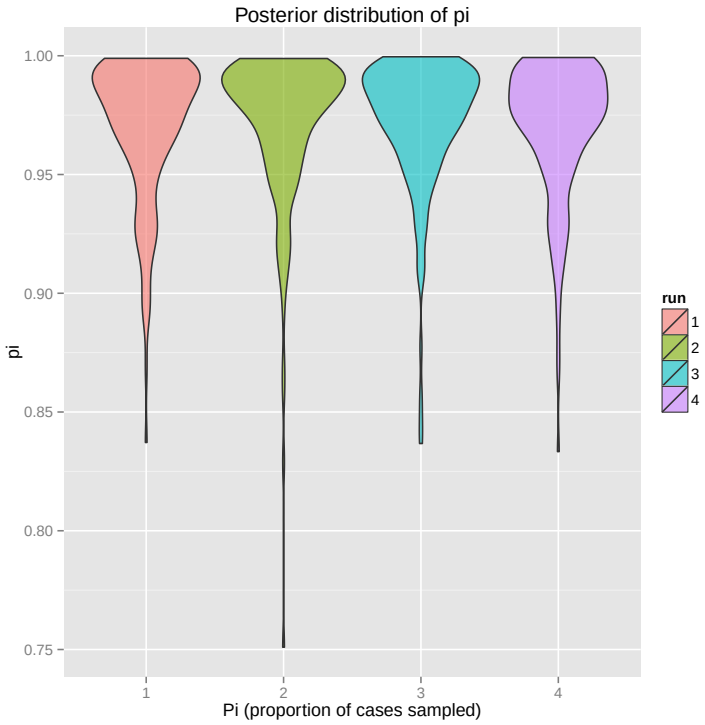
Using different boxplots for different runs:

```
p + geom_boxplot(aes(x=run, y=pi, fill=run),alpha=.6) +
    geom_jitter(aes(x=run, y=pi, col=run),alpha=.4)
```



Same idea, but using violinplots:

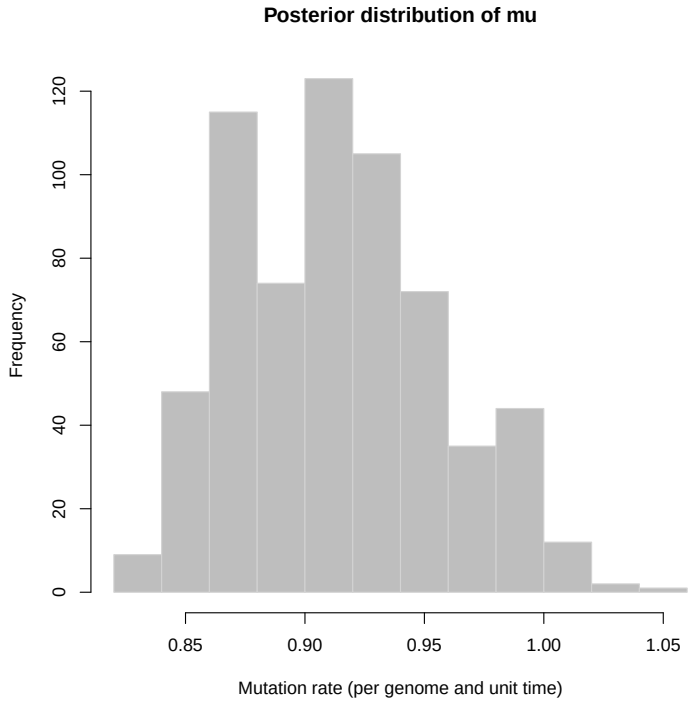
```
p + geom_violin(aes(x=run, y=pi, fill=run),alpha=.6)
```



2.4 Mutation rates

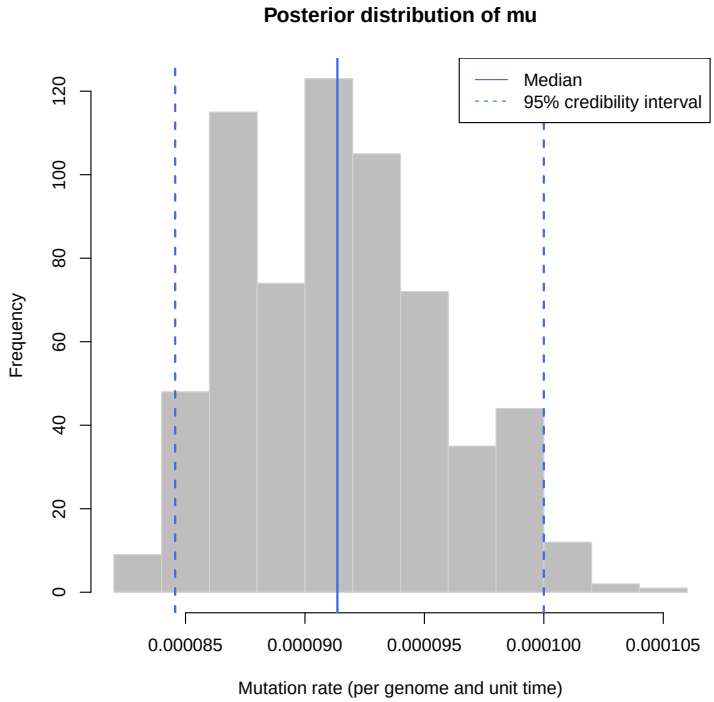
As mentioned before, mutation rates in *outbreaker*'s model are expressed per generation of infection. However, mutation rates per unit of time are biologically easier to interpret. These can be obtained using `get.mu`:

```
mu <- get.mu(res, burnin=2e4)
summary(mu)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.827   0.877   0.914   0.914   0.944   1.060

hist(mu, col="grey",border="lightgrey", xlab="Mutation rate (per genome and unit time)",
     main="Posterior distribution of mu")
```



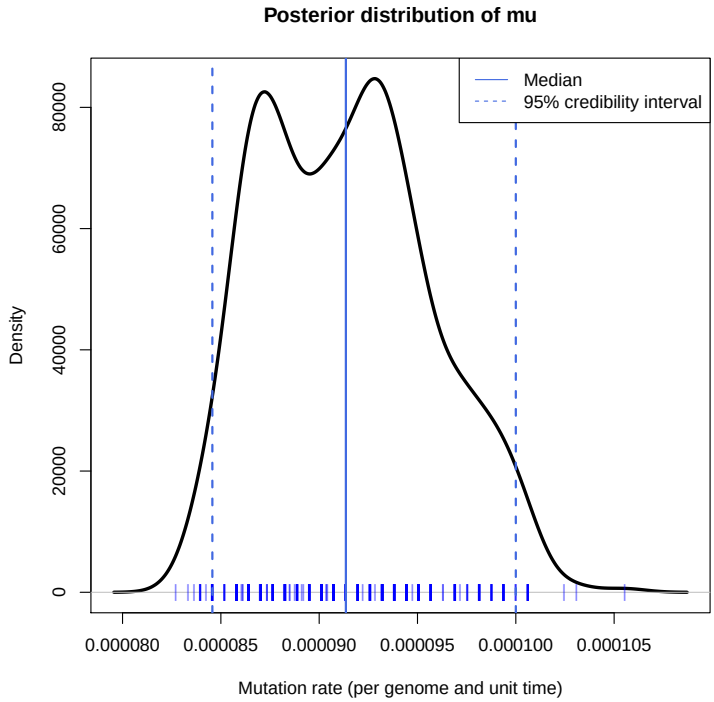
Re-expressing the rates per nucleotide, and adding the media and 95%credibility interval to the histogram:

```
mu <- get.mu(res, burnin=2e4, genome.size=ncol(dat$dna))
summary(mu)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 8.27e-05 8.77e-05 9.14e-05 9.14e-05 9.44e-05 1.06e-04

hist(mu, col="grey",border="lightgrey", xlab="Mutation rate (per genome and unit time)",
     main="Posterior distribution of mu")
abline(v=quantile(mu, c(.025, .5, .975)), lty=c(2,1,2), lwd=2, col="royalblue")
legend("topright", lty=c(1,2), leg=c("Median","95% credibility interval"),
       col="royalblue", bg="white")
```



Same idea, using a one-dimensional density estimation:

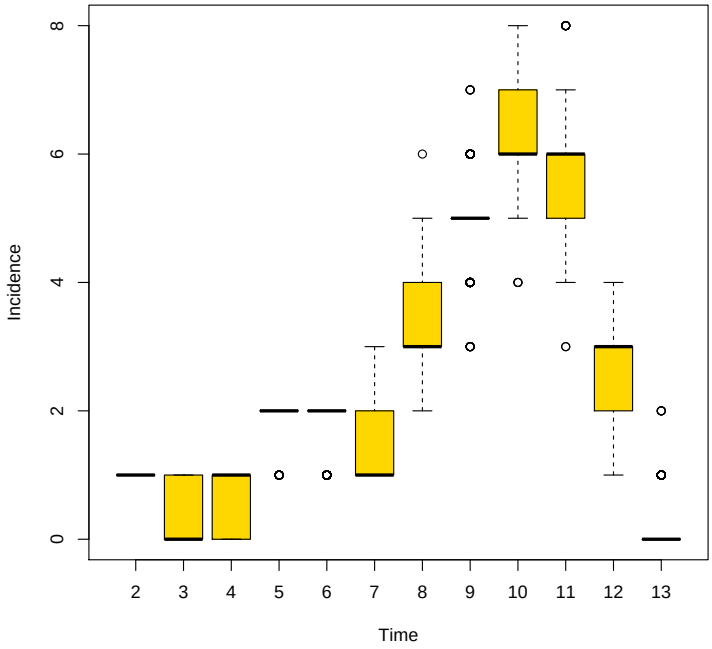
```
plot(density(mu), xlab="Mutation rate (per genome and unit time)",
     main="Posterior distribution of mu", lwd=3)
points(jitter(mu), rep(0, length(mu)), pch="|", col=transp("blue"))
abline(v=quantile(mu, c(.025, .5, .975)), lty=c(2,1,2), lwd=2, col="royalblue")
legend("topright", lty=c(1,2), leg=c("Median","95% credibility interval"),
       col="royalblue", bg="white")
```



2.5 Incidence and reproduction numbers

Incidence curves and effective reproduction numbers can be derived from the results of *outbreaker*. Incidence curves can be obtained and visualized using `get.incid`:

```
incid <- get.incid(res, burnin=2e4)
```




```
class(incid)
```

```
## [1] "matrix"
```

```
dim(incid)
```

```
## [1]  12 640
```

```
incid[,1:10]
```

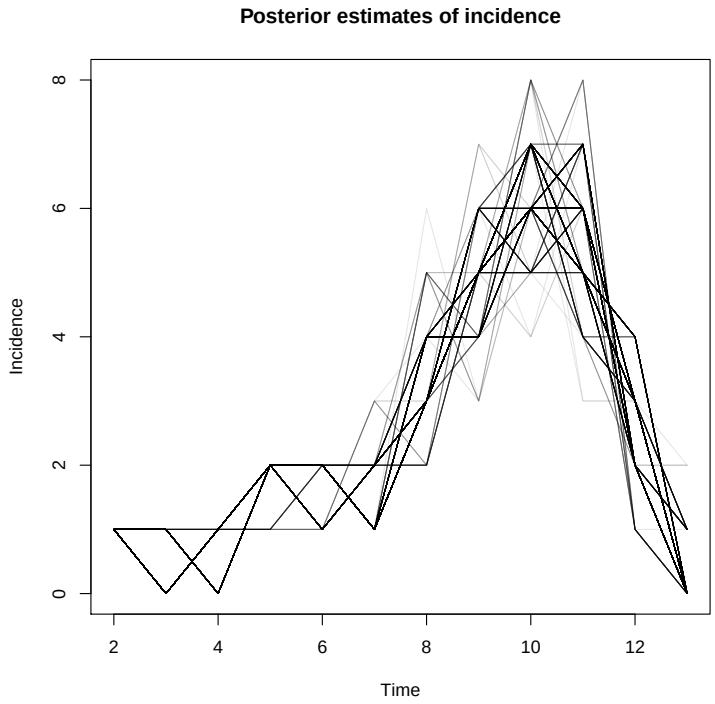
```
##    init
## 2     1 1 1 1 1 1 1 1 1 1
## 3     0 0 0 1 0 0 1 0 0 0
## 4     1 1 1 0 1 1 0 1 1 1
## 5     2 2 2 2 2 2 2 2 2 2
## 6     2 2 2 2 2 2 2 2 2 2
## 7     1 1 1 1 1 1 1 1 1 1
## 8     4 3 3 3 4 4 3 3 4 3
## 9     4 5 6 6 4 4 5 7 4 6
## 10    7 7 5 6 6 6 6 6 6 5
## 11    6 5 6 5 7 7 6 4 7 7
## 12    2 3 3 3 2 2 3 3 2 2
## 13    0 0 0 0 0 0 0 0 0 0
```

Each column in `incid` is a different realization of an incidence curve corresponding to one step of the MCMC. Other graphical options are available:

```
args(get.incid)
```

```
## function (x, burnin = 20000, plot = TRUE, type = c("boxplot",
##     "lines"), lines = FALSE, fill.col = "gold", lines.col = transp("grey"),
```

```
##     ...)
## NULL

incid <- get.incid(res, type="lines",lines.col=transp("black",.1))
title("Posterior estimates of incidence")
```

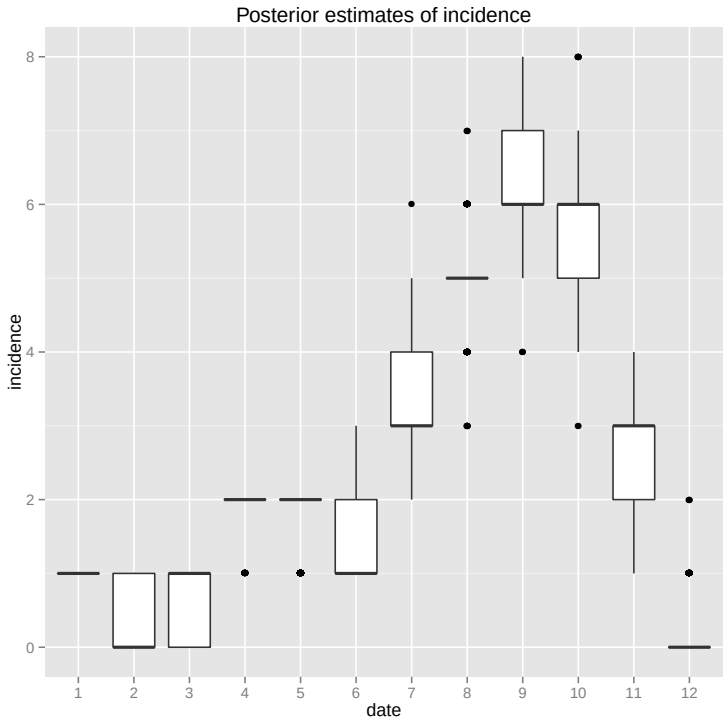


Because a lot of these trajectories overlap, visualizing them all is a bit tricky. *ggplot2* will be helpful here. We first reformat the data:

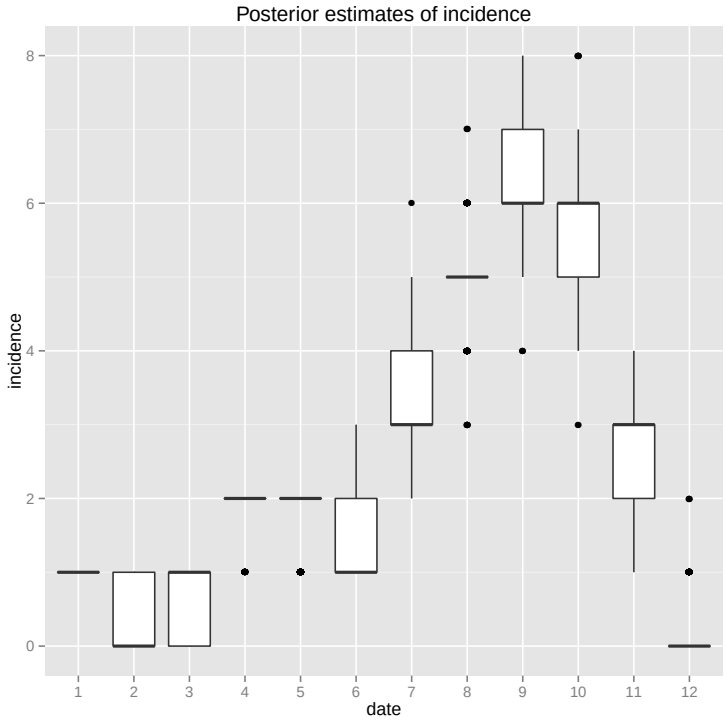
```
x <- data.frame(date=as.vector(row(incid)),
                step=as.vector(col(incid)),
                incidence=as.vector(incid))
head(x)

##   date step incidence
## 1    1    1         1
## 2    2    1         0
## 3    3    1         1
## 4    4    1         2
## 5    5    1         2
## 6    6    1         1

p <- ggplot(data=x, aes(x=date, y=incidence)) + labs(title="Posterior estimates of incidence")
```

This is for a basic boxplot:

```
p + geom_boxplot(aes(x=factor(date)))
```

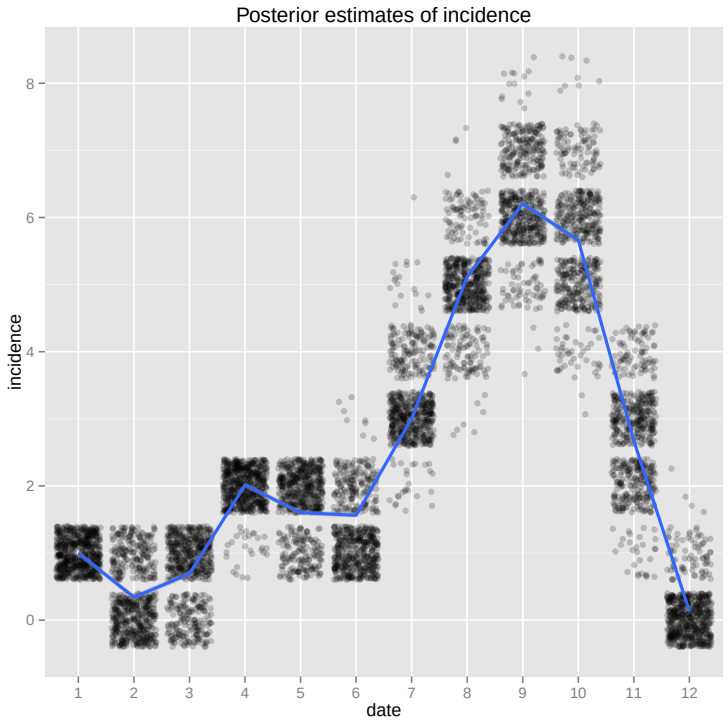



```
p + geom_boxplot(aes(x=factor(date)))
```



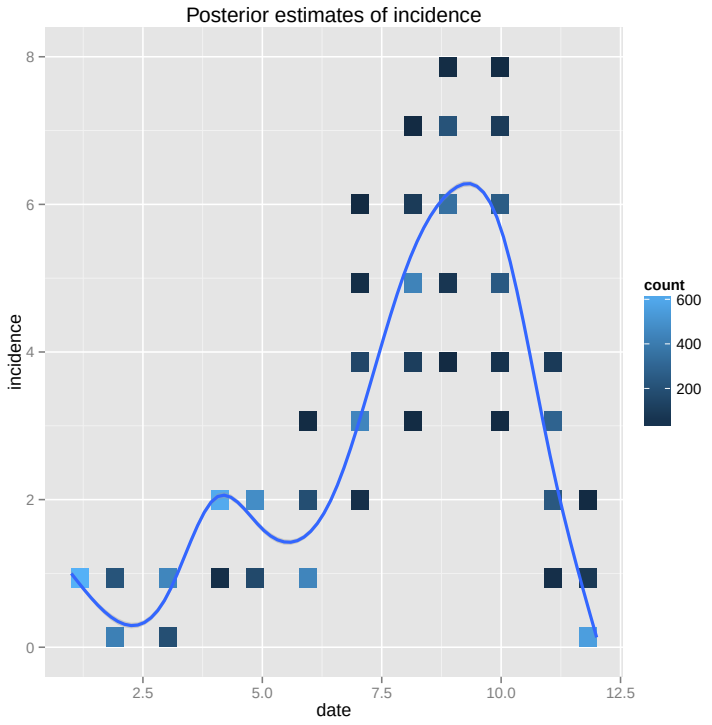
A noisified version with a smoother can be obtained using:

```
library(splines)
p + geom_jitter(aes(x=factor(date)), alpha=.2) + geom_smooth(method=lm, formula=y~ns(x,10), size=1)
```



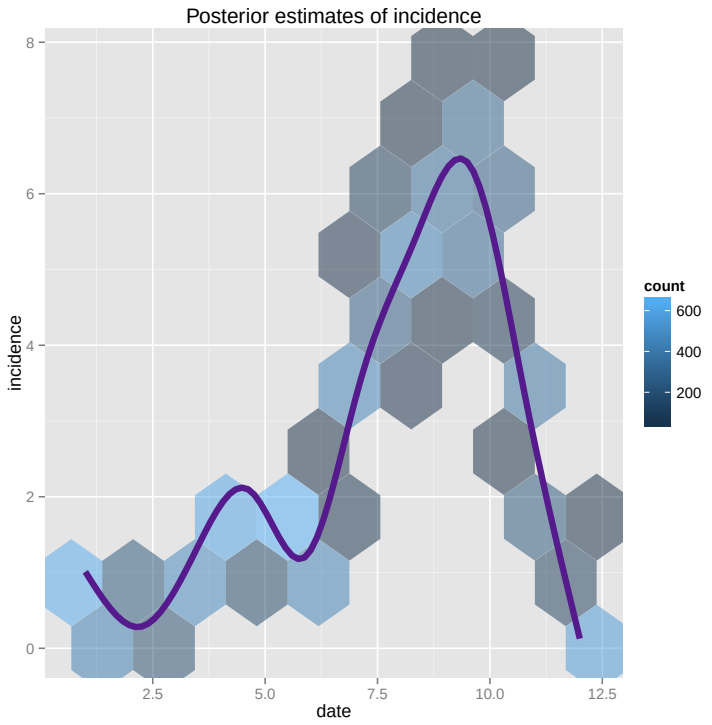
However, it adds artifactual variation and may be best avoided here.
Another interesting option is offered by some density-based graphics:

```
p + geom_bin2d() + geom_smooth(method=lm, formula=y~ns(x,10), size=1)
```



or perhaps even better:

```
p + geom_hex(bins=8, alpha=.5) + geom_smooth(colour="purple4",size=2)

## geom_smooth:  method="auto" and size of largest group is >=1000, so using gam with
formula:  y ~ s(x, bs = "cs").  Use 'method = x' to change the smoothing method.
```

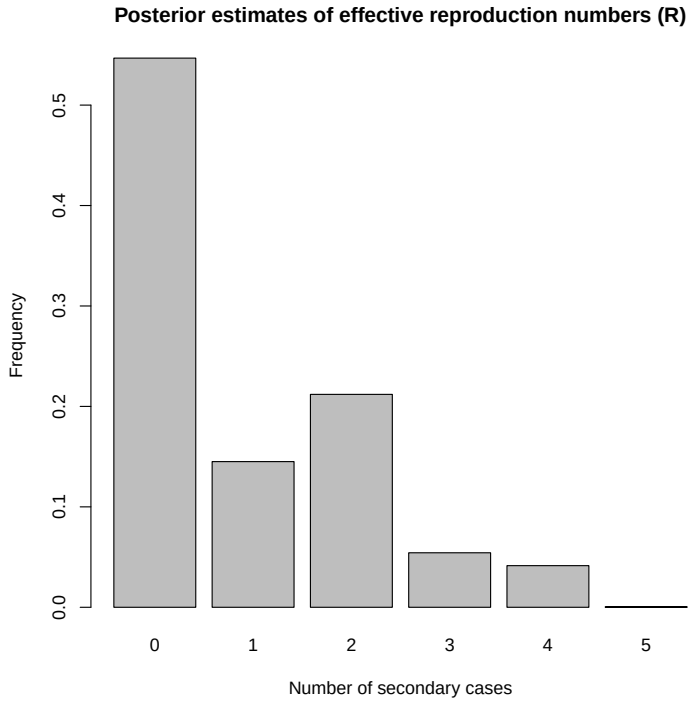


Effective reproduction numbers over time can be obtained using `get.Rt`, and given identical outputs to `get.incid`. Effective reproduction numbers per individuals can be obtained by `get.R`:

```
R <- get.R(res)
class(R)

## [1] "matrix"

dim(R)

## [1] 640  30

R[1:6,1:15]

##    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
## 42 1 1 1 2 3 2 3 2 2  2  2  0  4  0  0
## 43 1 1 1 2 4 2 3 0 2  2  2  1  4  0  0
## 44 1 1 1 2 3 2 3 2 2  2  2  0  4  0  0
## 45 1 1 1 2 5 2 3 0 2  1  2  0  4  0  0
## 46 1 1 1 2 4 2 3 1 2  2  2  0  4  0  0
## 47 1 1 1 2 4 2 3 0 2  2  2  1  4  0  0
```

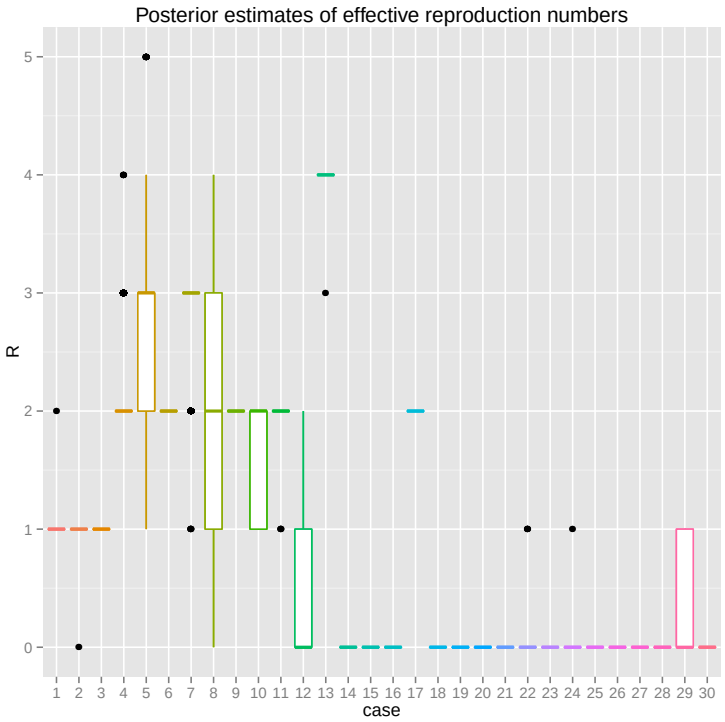
`R` is a matrix of effective reproduction numbers with one column per individual and one row for each retained step of the MCMC. This information can be pooled to get a rough idea of the overall distribution of R_i:

```
table(R)

## R
##     0     1     2     3     4     5
## 10498  2785  4070  1042   796     9

table(R)/length(R)

## R
##         0         1         2         3         4         5
## 0.5467708 0.1450521 0.2119792 0.0542708 0.0414583 0.0004687

barplot(table(R)/length(R), xlab="Number of secondary cases",
        main="Posterior estimates of effective reproduction numbers (R)", ylab="Frequency")
```



And we can use *ggplot2* to visualize the individual distributions:

```
x <- data.frame(case=factor(as.vector(col(R)), levels=as.character(1:30)),R=as.vector(R))
head(x)

##   case R
## 1    1 1
## 2    1 1
## 3    1 1
## 4    1 1
## 5    1 1
## 6    1 1

tail(x)

##       case R
## 19195   30 0
## 19196   30 0
## 19197   30 0
## 19198   30 0
## 19199   30 0
## 19200   30 0

p <- ggplot(data=x, aes(x=case, y=R)) + geom_boxplot(aes(colour=case))
p <- p + geom_boxplot(aes(colour=case)) + guides(colour=FALSE)
p + labs(title="Posterior estimates of effective reproduction numbers")
```


Posterior estimates of effective reproduction numbers
R
case
0
1
2
3
4
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

References

[1] Jombart T, Cori A, Didelot X, Cauchemez S, Fraser C, and Ferguson N. Bayesian reconstruction of disease outbreaks by combining epidemiologic and genomic data. *PLoS Computational Biology*, accepted.

[2] Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009.