How to reconstruct an outbreak and forwards simulate with BORIS.

Simon Firestone

Faculty of Veterinary and Agricultural Sciences

The University of Melbourne, Parkville Victoria 3010 Australia

simon.firestone@unimelb.edu.au

25/11/2019

Contents

Outbreak reconstruction and inference with BORIS

In this exercise, we will:

1. Prepare the inputs for outbreak reconstruction and inference
2. Run the reconstruction and inference
3. Inspect diagnostics for the MCMC chain
4. Produce some relevant outputs

Some of the guidance points to help pages for functions and datasets in the BORIS package. To access help on any functions or datasets in the BORIS package, type the following into the R console:

```
#load the library or install it if it is missing
require(BORIS)

#load the help page for the package
help(package="BORIS")
```

For any issues with installation see the guide available here.

Overview of inputs

The following inputs are required to run an outbreak reconstruction and inference:

1. `covariates`: farm covariate data inputed as specified in the help for `data(infer.epi.input)`. We will prepare this file from observed field data.
2. `movements`: Animal movement or contact-tracing data inputed as specified in the help for `data(infer.moves.input)`. If the option for considering movements in the analysis is disabled, i.e. `opt_mov = 0`, then this input will be ignored.
3. `sequences`: A `*.fasta` file with sequence data for each farm. For those farms that have not been sampled a sequence is entered with missing data.

Further sets of parameters are entered specific to the model run, as follows:

1. `parsAux`: Parameters for controlling the MCMC implementation.
2. `keyInits`: Key initial values for inferred parameters.
3. `priors`: Prior settings for the inferred parameters.
4. `scalingFactors`: Scaling factors for the inferred parameters.
5. `seed`: An integer seed for the random number generator.
6. `accTable`: Known sources for each individual for simulated outbreaks. If unknown, i.e. inferring for data from an observed outbreak, then enter `9999` for each source.

Scenario

An outbreak of foot-and-mouth disease (FMD) has occurred in a previously free region. There are 100 known infected farms. We would like to reconstruct the outbreak and infer the transmission network between these farms (i.e. 'who infected whom') and additional important parameters including: the timing of exposure and infectiousness of animals on each farm, the spatial risk around infected farms, the duration of key epidemiological periods (latent and infectious periods), risk of infection depending on certain farm types, and transmission parameters for forwards simulation.

Data has been collected from every infected premises (IP) on:

- its location
- the timing of earliest onset of clinical signs in any animals on the farm
- the timing of sampling that has led to genomic sequence data
- the farm type
- the number of susceptible animals on the farm
- contact-traced movements between some of the farms

Preparing inputs

`covariates`

We will prepare the `covariates` and `movements` data from the file `outbreak_inputs.xlsx`.

First open and inspect this file in Excel, then load it into `R` using the library `readxl` as follows:

```
#load the library or install it if it is missing
require(readxl)
#> Loading required package: readxl

#check which sheets are in the input file
excel_sheets('outbreak_inputs.xlsx')
#> [1] "covariates" "movements"  "sources"

#load the covariate data into a new object call `cov`
cov <- as.data.frame(read_excel('outbreak_inputs.xlsx', sheet = 'covariates'))

#inspect the top few rows of `cov`
head(cov)
#>   k   coor_x    coor_y t_o t_s t_r ftype herdn
#> 1 0 146.0816 -38.32329   4   9  29     1  3209
#> 2 1 146.0831 -38.32280  13  20  27     2  1705
#> 3 2 146.0831 -38.32280  98  97 100     0   133
#> 4 3 146.0817 -38.32212  15  15  35     0   213
#> 5 4 146.0820 -38.32190  12  15  37     0   127
#> 6 5 146.0814 -38.32298  10  12  26     2     6

#inspect the last few rows of `cov`
tail(cov)
#>       k  coor_x   coor_y     t_o     t_s t_r ftype herdn
#> 145 144 145.877 -38.3403 9000000 9000000 100     2     5
#> 146 145 145.880 -38.4439      72      74  79     0   425
#> 147 146 146.192 -38.3988 9000000 9000000 100     0   308
#> 148 147 145.916 -38.2808 9000000 9000000 100     0    92
#> 149 148 145.872 -38.2744 9000000 9000000 100     1     5
#> 150 149 146.242 -38.4174 9000000 9000000 100     0     6
```

The `covariates` data is structured as one row per farm, with the following fields:

`k`:: A unique identifier for each farm. Starts at zero given C++ indexing system.

`coor_x`: x coordinate of the farm's centroid, e.g., Longitude in decimal degrees using WGS84 datum.

`coor_y`: y coordinate of the farm's centroid, e.g., Latitude in decimal degrees using WGS84 datum.

`t_o`: the timing of earliest onset of clinical signs in any animals on the farm, measured in days from some arbitrary origin (`day 0`). Left as the unassigned value (default 9e06), for uninfected farms or farms where clinical onset was not observed.

`t_s`: the timing of sampling that has led to genomic sequence data. Left as the unassigned value (default 9e06), for uninfected farms or farms where sampling did not occur.

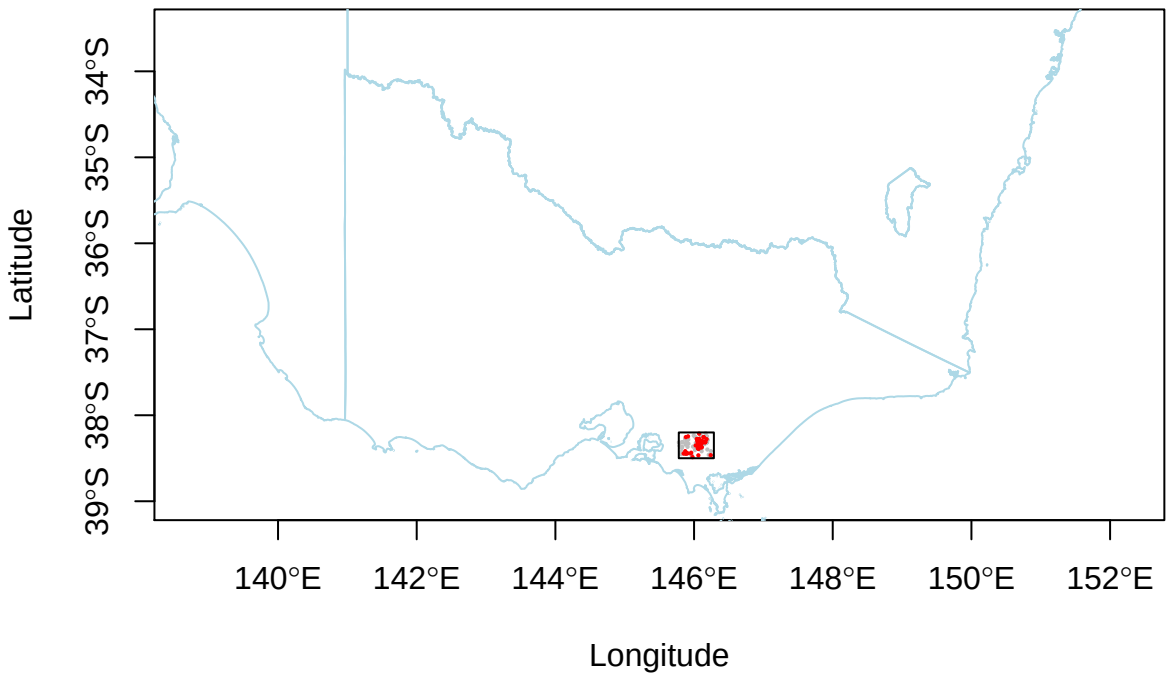
`t_r`: Day removed, recovered or culled, if known or the unassigned value (default 9e06) if unknown.

`ftype`: A 3-level categorical variable used to represent farm type, here, in terms of the predominant susceptible species held, i.e. ftype = 0 if predominant species is cattle, ftype = 1 if predominant species is pigs, ftype = 2 if predominant species is sheep/other. This variable is later converted into an indicator variable.

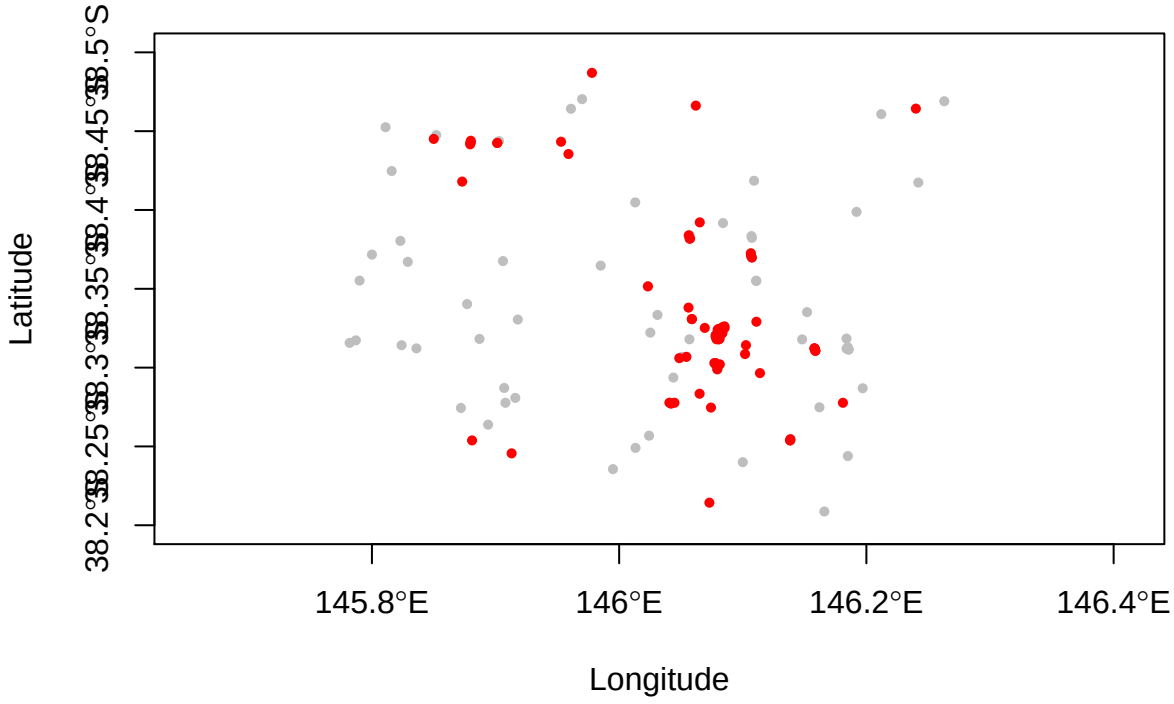
`herdn`: Number of susceptible animals held on the farm at the start of the outbreak.

Let's map these farms by their infection status and display an epidemic curve:

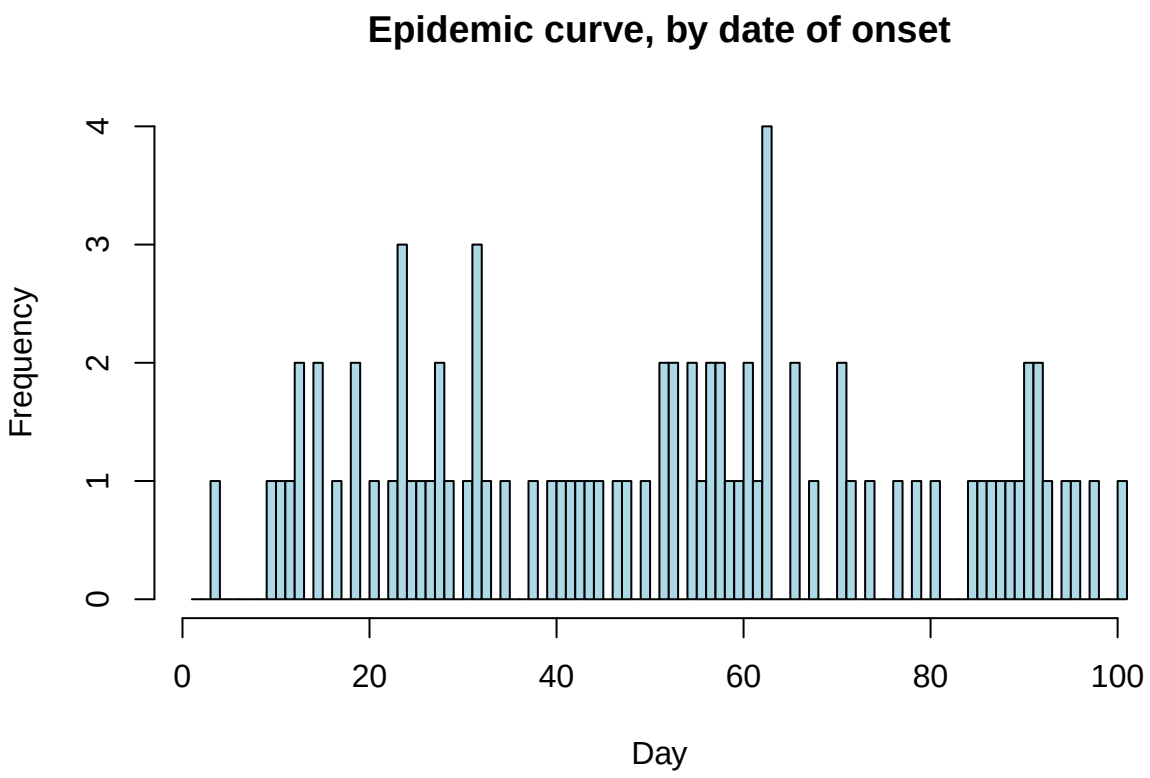
```
#first, identify which farm have been infected premises (IPs) at some stage
which(cov$t_o < 9e6)
#>  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  18  19  20  21
#> [18]  22  23  24  26  27  28  29  30  33  34  35  36  37  38  39  41  43
#> [35]  44  45  46  48  49  51  53  54  55  57  58  62  63  66  71  72  73
#> [52]  75  77  78  81  82  83  84  86  87  88  89  90  91  92  93  95  96
#> [69]  97  98 104 107 108 117 118 120 122 123 128 130 133 134 146
#store this in an object so it can be used to select these farms
ips<-which(cov$t_o < 9e6)
#and identify the non-ips
nips<-which(cov$t_o == 9e6)

#load library for handling spatial data
library(rgdal)
#load the Ausrtalian boundary in GDA94 coordinate reference system:
aus.shp <- readOGR("AU_geotopo-gda94.shp")
#> OGR data source with driver: ESRI Shapefile
#> Source: "C:\Temp\AU_geotopo-gda94.shp", layer: "AU_geotopo-gda94"
#> with 6121 features
#> It has 14 fields

#plot a map of the farms, by infection status, with the study area bounding box
plot(aus.shp, bor='lightblue', axes = TRUE, xlab = "Longitude", ylab = "Latitude", xlim=c(140.5,150.5),
points(cov$coor_x[nips], cov$coor_y[nips], pch = 16, col = "grey", cex=0.3)
points(cov$coor_x[ips], cov$coor_y[ips], pch = 16, col = "red", cex=0.3)
#plot an extent window
rect(xleft=145.7795, ybottom=-38.20, xright=146.2856, ytop=-38.5, col="transparent", border='black')
```




```
#plot a map zoomed in on the extent window
plot(aus.shp, bor='lightblue', axes = TRUE, xlab = "Longitude", ylab = "Latitude", xlim=c(145.7795,146.1
points(cov$coor_x[nips], cov$coor_y[nips], pch = 16, col = "grey", cex=0.7)
points(cov$coor_x[ips], cov$coor_y[ips], pch = 16, col = "red", cex=0.7)
```




```
#display an epidemic curve
tmp<-cov$t_o[ips]
hist(tmp, breaks = 1:max(tmp), col='lightblue',
     main='Epidemic curve, by date of onset', xlab='Day')
```



Typically, when working with empirical data from an outbreak that has been detected, the day of exposure (`t_e`) and onset of infectiousness (`t_i`) are unobserved and initial values have to be estimated by backwards calculation from the day clinical signs were first observed in an animal on a farm (`t_o`). The values of `t_e` and `t_i` are then updated by the Bayesian MCMC process.

This can be done stochastically, using Gamma distributions to represent the incubation period (`t_e` to `t_o`) and the latent period (`t_e` to `t_i`).

For foot-and-mouth disease (FMD), Alexanderson et al. (2003) estimated these key epidemiological periods to be:

- incubation period: 1-14 days, typically 2-5 days.
- the most likely period of infectiousness for cattle is between 1 and 5 days.
- the infectious period usually begins up to 1 day prior to or at the appearance of clinical signs. Noting that virus can be detected in milk up to 4 days before the appearance of clinical signs.

See here for further details, and an example of backwards calculations from clinical signs typical for FMD.

In this exercise, for simplicity, we will assume for the initial values that:

- first exposure on each farm (`t_e`) occurred 3 days before the earliest onset of clinical signs (`t_o`), i.e. an incubation period of 3 days.
- exposure (`t_e`) occurred 48 hours before the onset of infectiousness (t_i), , i.e. a latent period of 2 days.
- the duration of infectiousness is unknown, and this is no issue for initialisation.

The following code prepares the `covariates` input file as required for the function `BORIS::infer()`:

```
# produce the required initial values for the temporal fields.

#assume that exposure has occurred 3 days before earliest onset of clinical signs:
cov$t_e <- cov$t_o - 3
#for non-IPs keep this as the unassigned value
cov$t_e[nips] <- 9e6

#assume that onset of infectiousness has occurred 48 hours after exposure:
cov$t_i <- cov$t_e + 2
cov$t_i[nips] <- 9e6

#store the timing of sampling separately for now
t_s <- cov$t_s

#keep just the rest of the desired columns, re-ordered as required
cov<-cov[,c('k','coor_x','coor_y','t_e','t_i','t_r','ftype','herdn')]
head(cov)
#>   k   coor_x    coor_y t_e t_i t_r ftype herdn
#> 1 0 146.0816 -38.32329   1   3  29     1  3209
#> 2 1 146.0831 -38.32280  10  12  27     2  1705
#> 3 2 146.0831 -38.32280  95  97 100     0   133
#> 4 3 146.0817 -38.32212  12  14  35     0   213
#> 5 4 146.0820 -38.32190   9  11  37     0   127
#> 6 5 146.0814 -38.32298   7   9  26     2     6
```

For each individual propose an initial source for the tree used to initialised the MCMC inference. This could be assigned manually based on available information and/or expert opinion. Or, as in the below example randomly sampled from those farms with earlier initial exposure times than each farm of interest.

```
cov$initial_source<-NA

for (i in 1:nrow(cov)){

  cov$k[i]

  #identify the pool of possible sources with earlier initial exposure times
  id <- which(cov$t_e<cov$t_e[i] & cov$t_r>cov$t_e[i])
  pool_source <- cov$k[id]

  #randomly assign an infection source is there are any in the pool
  if (length(pool_source)>1) cov$initial_source[i] <- sample(pool_source,1)
  if (length(pool_source)==1) cov$initial_source[i] <- pool_source[1]

  #otherwise assign 9999 as the initial source, to indicate that it is either external to the dataset o
  if (length(pool_source)<1) cov$initial_source[i] <- 9999

}

head(cov)
#>   k   coor_x    coor_y t_e t_i t_r ftype herdn initial_source
```

```
#> 1 0 146.0816 -38.32329   1   3  29     1  3209          9999
#> 2 1 146.0831 -38.32280  10  12  27     2  1705             0
#> 3 2 146.0831 -38.32280  95  97 100     0   133            88
#> 4 3 146.0817 -38.32212  12  14  35     0   213             4
#> 5 4 146.0820 -38.32190   9  11  37     0   127            65
#> 6 5 146.0814 -38.32298   7   9  26     2     6             0
```

From this process it has been assumed that the initial source for farm `id` = 0 is external to the data (i.e. 9999). It could be a seed for this cluster, or its source may have not been sampled (though here we are assuming all have been sampled). Check for yourself if any of the other IPs are suggested as having initial sources external to the data.

The `covariates` data are now ready.

movements

The movements data are on the 2nd sheet in the file `outbreak_inputs.xlsx`, and are formatted already as required. Load these as follows:

```
#check which sheets are in the input file
excel_sheets('outbreak_inputs.xlsx')
#> [1] "covariates" "movements"  "sources"

#load the covariate data into a new object call `movs`
movs <- as.data.frame(read_excel('outbreak_inputs.xlsx', sheet = 2))

#inspect the top few rows of `movs`
head(movs)
#>   from_k to_k t_m
#> 1      0    1   9
#> 2      0    2   9
#> 3      0    3   9
#> 4      0    4   9
#> 5      0    5   9
#> 6      0    6   9
```

The `movements` data is structured as one row per movement, with the following fields:

`from_k`: the unique identifier of the source farm for the animals moved. Starts at zero given C++ indexing system. They refer to the unique farm ids in the column `id` in the `covariate` data.

`to_k`: the unique identifier of the destination farm for the animals moved. Starts at zero given C++ indexing system.

`t_m`: day that the movement occurred.

sequences

The sequences data are provided in fasta format as required in the file `outbreak_seqs.fasta`. These can be opened in all common phylogenetic programs like MEGA. Genomic data can also be viewed, manipulated and written to other file formats with the R library ape.

Please put this file into its own directory, preferrably wherever you are working, i.e., `C:\Temp\gen_inputs`. The function `infer()` requires the full path to be specified to the directory and that it contains just a single *.fasta file.

parsAux

The options for configuring the MCMC implementation are setup as follows.

```
pars.aux <- data.frame('n' = nrow(cov),
                       'kernel_type' = 'power_law',
                       'coord_type' = 'longlat',
                       't_max' = 100,
                       'unassigned_time' =  9e+6,
                       'processes' = 1,
                       'n_seq' = 5,
                       'n_base' = 7667,
                       'n_iterations' = 1e5,
                       'n_frequ' = 10,
                       'n_output_source' = 1,
                       'n_output_gm' = 1000,
                       'n_cout' = 10,
                       'opt_latgamma' = 1,
                       'opt_k80' = 1,
                       'opt_betaij' = 1,
                       'opt_ti_update' = 1,
                       'opt_mov' = 0,
                       stringsAsFactors = F)

#once `t_max` is set, check that all recovery times are < t_max, and right censor as required
cov$t_r[cov$t_r>pars.aux$t_max]<-pars.aux$t_max
```

For details of these parameters see the help for `infer.param.aux`.

```
require(BORIS)
help(infer.param.aux)
```

In this example, `opt_mov` is set to zero. We aren't running the inference incorporating movement data, as it slows the process.

keyInits

The initial values for key inferred parameters are setup as follows. For details see the help for `infer.param.key`.

```
para.key.inits <- data.frame('alpha' = 2e-4,
                             'beta' = 0.01,
                             'lat_mu' = 5,
                             'lat_var' = 1,
```

```
                                'c' = 4,
                                'd' = 1,
                                'k_1' = 10,
                                'mu_1' = 1e-05,
                                'mu_2' = 5e-06,
                                'p_ber' = 0.2,
                                'phi_inf1' = 1,
                                'phi_inf2' = 1,
                                'rho_susc1' = 1,
                                'rho_susc2' = 1,
                                'nu_inf' = 0,
                                'tau_susc'= 0,
                                'beta_m'= 1)
```

priors

The prior information for inferred parameters are setup as follows. For details see the help for `infer.param.priors`.

```
para.priors <- data.frame('t_range' = 7,
                          't_back' = 21,
                          't_bound_hi' = 10,
                          'rate_exp_prior' = 0.001,
                          'ind_n_base_part' = 0,
                          'n_base_part' = 1000,
                          'alpha_hi' = 0.1,
                          'beta_hi' = 50,
                          'mu_lat_hi' = 50,
                          'var_lat_lo' = 0.1,
                          'var_lat_hi' = 50,
                          'c_hi' = 100,
                          'd_hi' = 100,
                          'k_1_hi' = 100,
                          'mu_1_hi' = 0.1,
                          'mu_2_hi' = 0.1,
                          'p_ber_hi' = 1.0,
                          'phi_inf1_hi' = 500,
                          'phi_inf2_hi' = 500,
                          'rho_susc1_hi' = 500,
                          'rho_susc2_hi' = 500,
                          'nu_inf_lo' = 0,
                          'nu_inf_hi' = 1,
                          'tau_susc_lo' = 0,
                          'tau_susc_hi' = 1,
                          'beta_m_hi' = 5,
                          'trace_window' = 20)
```

scalingFactors

The scaling factors (otherwise know as operators or proposal distances) for inferred parameters are setup as follows. For details see the help for `infer.param.sf`. The default is 1. The scaling factors are used to

'tune' the MCMC so it searchs the multidimensional parameter space efficiently. Increase the scaling factor to increase the proposal distance, for instance if the parameter is wandering about and accepting too often (i.e. if the proposal distance is too low). Decrease the scaling factor to decrease the proposal distance, for instance if the parameter is not being accepted often enough (i.e. if the proposal distance is too high and its trace looks like a 'Manhattan skyline'). Set the scaling factor to zero to fix a parameter at its initialising value (so that it doesn't update).

```
para.sf <- data.frame('alpha_sf' = 0.001,
                      'beta_sf' = 1,
                      'mu_lat_sf'   = 1,
                      'var_lat_sf' = 1,
                      'c_sf' = 1,
                      'd_sf' = 0.5,
                      'k_1_sf' = 1,
                      'mu_1_sf' = 5e-5,
                      'mu_2_sf' = 5e-6,
                      'p_ber_sf' = 0.05,
                      'phi_inf1_sf' = 1.5,
                      'phi_inf2_sf' = 1,
                      'rho_susc1_sf' = 1,
                      'rho_susc2_sf' = 1,
                      'nu_inf_sf' = 1,
                      'tau_susc_sf' = 1,
                      'beta_m_sf' = 0)
```

`accTable`

As the accuracy table (`accTable`) for evaluating how well the infer is performing, we will use the known infected sources from the simulation that originally generated these data (the outbreak is in Australia after all). When inferring for an actual outbreak, it will be unknown which is the true source for each infected individual. In that case, enter `9999` for all infected sources with the code `accTable = rep(9999, pars.aux$n)`.

```
#as this outbreak was simulated (it is in Australia!) we know the sources
accTable<-as.numeric(as.data.frame(read_excel('outbreak_inputs.xlsx',
                      sheet = "sources",col_names = F))[,1])
#> New names:
#> * `` -> ...1
```

Running the inference

Here we just run one MCMC chain for a very small number of iterations (for demonstration purposes only), whereas in practice multiple chains must be run and inspected appropriately (i.e. for burn-in, convergence and serial autocorrelation) before any inferences are made on the transmission tree or any inferred parameters.

An integer seed is set for the random number generator so the MCMC run can be reproduced. We'll just use 1 in this example. It should be set randomly for each chain in acutal analyses.

For demonstration purposes only, we will set the desired number of iterations to 1000. In practice this should be set based on assessment of convergence of multiple chains, and from experience with such complex dynamix models, typically multiple chains of >100,000 are required, even millions of iterations, to achieve convergence and an appropriate effective sample size.

```
#load the BORIS library
require(BORIS)

#set the number of iterations
pars.aux$n_iterations = 1000

#run the inference, storing the output in the object infer.out
#the input and output directories need to be specified
#if they do not exist they will be created
infer.out<-infer(covariates = cov,
                 moves.inputs = movs,
                 parsAux = pars.aux,
                 keyInits = para.key.inits,
                 priors = para.priors,
                 scalingFactors = para.sf,
                 seed = 1,
                 accTable = accTable,
                 t.sample = t_s,
                 inputPath = "./inputs",
                 outputPath = "./outputs",
                 dnaPath = "./gen_inputs")
```

Multiple chains can be run:

1. on a single computer: by saving the output from the first run to a new directory, because when `infer()` starts it deletes everything in the input and output directories. Then running again with a different seed.
2. on a parallel cluster that uses the Slurm Workload Manager or similar: adapt the following code, then use `.seed` as the `seed` argument for the function `infer()`:

```
#Only run this code as part of a Rscript on a SLURM-based cluster:
task.id <- as.numeric(Sys.getenv('SLURM_ARRAY_TASK_ID'))
task.id

.seed <- c(1, 102, 1003, 10004)[task.id]
.seed
```

```
infer.out<-infer(covariates = cov,
                 moves.inputs = movs,
                 parsAux = pars.aux,
                 keyInits = para.key.inits,
                 priors = para.priors,
                 scalingFactors = para.sf,
                 seed = 1,
                 accTable = accTable,
                 t.sample = t_s,
                 inputPath = "./inputs",
                 outputPath = "./outputs",
                 dnaPath = "./gen_inputs")
```

Inspecting outputs

Following the MCMC run, it is important to check and exclude burn-in appropriately, and for appropriate mixing and convergence of multiple chains. Tracer is a great tool for rapidly evaluating the parameter output of chains, as stored in the file `parameters_current.log`. It allows rapid assessment of convergence of each parameter across multiple chains, amount of autocorrelation, effective sample sizes, and also reading off summary posterior estimates.

Install `Tracer` and open the output file you generated `parameters_current.log` and inspect the chain for each parameter.

Next, we will implement some similar analyses in `R` on longer chains that have been pre-run, then continue with code for extracting posterior estimates of the transmission tree and further inferred parameters for each individual.

In the last example, we just ran a single chain. The following uses example data from 2 chains of the same inference, each of length 10,000 MCMC cycles. Again, this is for illustative purposes only, in practice hundreds of thousands to millions of iterations are typically required as these are complex inferences.

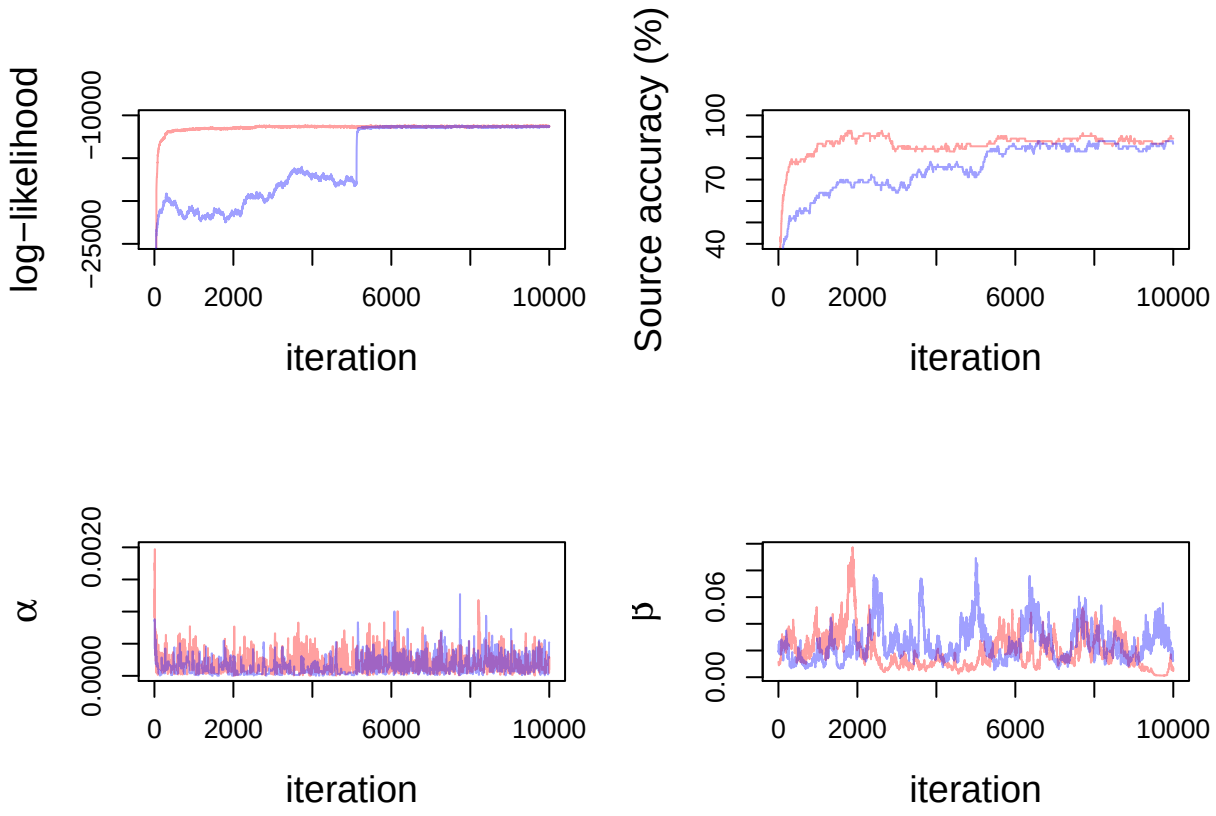
Inspecting the chains

Inspect the start of the chains to evaluate if and when convergence has occurred. Therefore, where to set burn-in, and if need be the amount of thinning.

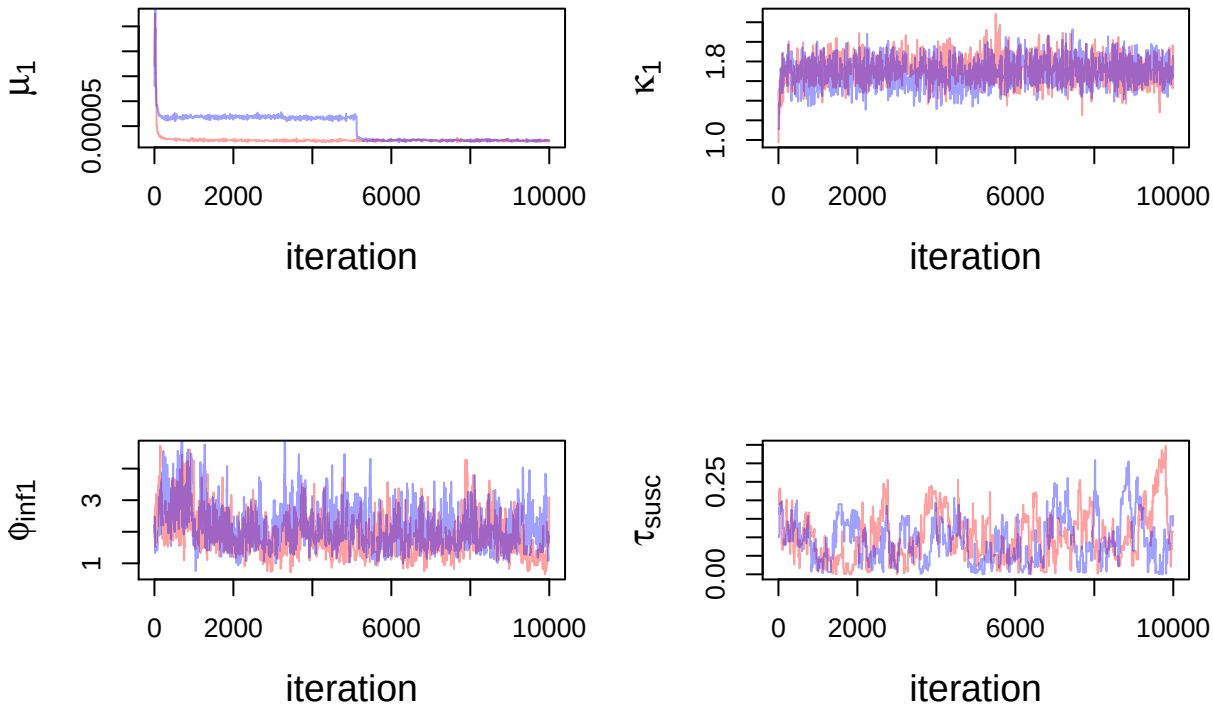
Pre-prepared parameter data for each chain are provided to input by reading in the tab delimited `run1_parameters_current.log` and `run2_parameters_current.log` files produced by independantly seeded `infer()` runs on the same input data.

```
#load pre-prepared infer output data from chain 1, run for 10000 iterations
paras1<-read.csv("run1_parameters_current.log", sep='\t')

#load pre-prepared infer output data from chain 2, run for 10000 iterations
paras2<-read.csv("run2_parameters_current.log", sep='\t')

#start with no-burnin to check
burnin=0; thin = 1; end=1e04
#store a vector of indices along the chain
x<-seq(burnin+1,end,thin)

#plot 1
par(mfrow=c(2,2)) #plot in 2 rows and 2 columns
plot(x, paras1$log_likelihood[x], type='l', col='#ff000060', ylim=c(-25e3,-10e3),
     xlab='iteration', ylab='log-likelihood', cex.lab=1.4)
lines(x, paras2$log_likelihood[x], col='#0000ff60')

plot(x, paras1$corr[x]/length(ips)*100, type='l', col='#ff000060', ylim=c(40,100),
     xlab='iteration', ylab='Source accuracy (%)', cex.lab=1.4)
lines(x, paras2$corr[x]/length(ips)*100, col='#0000ff60')

plot(x, paras1$alpha[x], type='l', col='#ff000060', ylim=c(0,0.002),
     xlab='iteration', ylab=expression(alpha), cex.lab=1.4)
lines(x, paras2$alpha[x], col='#0000ff60')
```

```
plot(x, paras1$beta[x], type='l', col='#ff000060',
     xlab='iteration', ylab=expression(beta), cex.lab=1.4)
lines(x, paras2$beta[x], col='#0000ff60')
```




```
#plot 2
par(mfrow=c(2,2))
plot(x, paras1$mu_1[x], type='l', col='#ff000060',
     xlab='iteration', ylab=expression(mu[1]), cex.lab=1.4)
lines(x, paras2$mu_1[x], col='#0000ff60')

plot(x, paras1$k_1[x], type='l', col='#ff000060',
     xlab='iteration', ylab=expression(kappa[1]), cex.lab=1.4)
lines(x, paras2$k_1[x], col='#0000ff60')

plot(x, paras1$phi_inf2[x], type='l', col='#ff000060',
     xlab='iteration', ylab=expression(phi[inf1]),
     cex.lab=1.4)
lines(x, paras2$phi_inf2[x], col='#0000ff60')

plot(x, paras1$tau_susc[x], type='l', col='#ff000060',
     xlab='iteration', ylab=expression(tau[susc]),
     cex.lab=1.4)
lines(x, paras2$tau_susc[x], col='#0000ff60')
```




```
par(mfrow=c(1,1)) #return the plotting parameters to 1 row and 1 column
```

You should always inspect all inferred parameters in all chains.

In this example, there are a number of really interesting patterns. Several parameters seem to converge after iteration 6000, and so too does the `log-likelihood` for these two chains. A shift to a more likely space could occur again, followed by convergence, so it would be hasty to assume convergence at 6000 for such a complex model.

High `accuracy` in identifying sources for IPs is achieved quickly, but continues to rise, and it would be wise to run these chains for longer.

The two chains for `alpha` appear to converge very quickly then mix well with most posterior samples with very low values of this the background transmission rate, which is bounded by zero.

The chains for `beta`, the secondary transmission rate, are looking like they might soon to converge, however there are patterns here that indicate **poor mixing** to be aware of if they persist longer into traces. In the first 2000 iterations, the red trace for `beta` wanders about, and it appears the rate of acceptance is too high. Throughout the first 10000 iterations for both chains of `beta` there seems to be a lot of serial autocorrelation. In this example, this is just because we haven't run enough iterations, and the chains can still be considered to be burning-in. Possible solutions to such poor mixing patterns could include running the MCMC for more iterations or altering the respective scaling factor (proposal distance). If the rate of acceptance is too high, then increase the scaling factor, so the MCMC chain proposes over greater distances. This will lead to more efficient traversal of the parameter space, and reduce the acceptance rate towards the optimum of 20%-30%.

The traces for `kappa_1`, the scaling parameter of the spatial kernel, look very nice. There is just a little evidence of autocorrelation, and convergence looks good and the scaling factor appears about right. These are close to the 'lazy furry caterpillars' that we're after.

The traces for `tau_susc`, the effect of herd size on susceptibility, have a mild 'Manhattan skyline' appearance. This suggests that the acceptance rate is too low and the chain is sticking a little on certain values over many iterations rather than mixing efficiently.

So, in this example, we will use a burn-in of 6000 iterations, because most parameters appear to be near convergence at this point. Typically at least 20% of each chain should be discarded as 'burn-in', and assessment should be made across all chains with consideration of effective sample size (aiming for >200 for all inferred parameters).

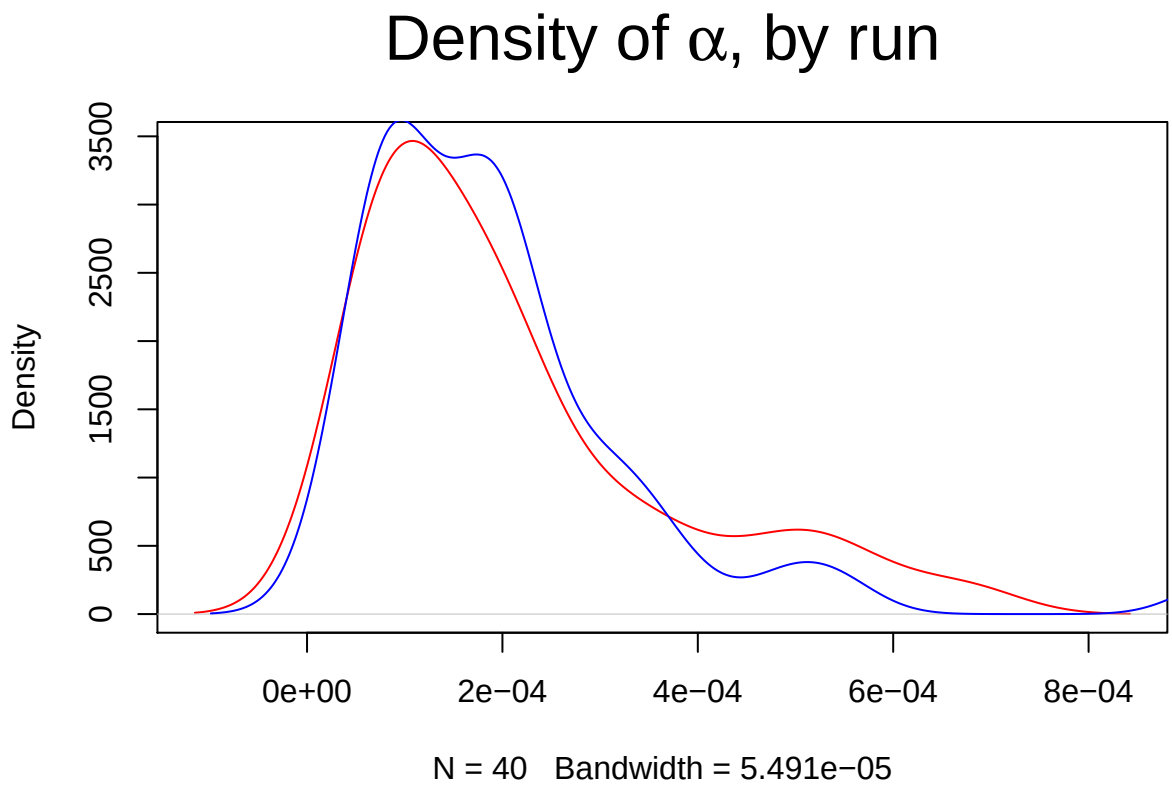
As there is a small amount of serial autocorrelation we will thin the estimates by a lag of 100. Formal estimation of autocorrelation can be undertaken with the `acf` function in `R`, or by checking the auto-correlation time (ACT) in Tracer, which shows the number of steps that are needed before the chain 'forgets' where it started, and is a suggested amount of thinning for the parameter.

The posterior parameter estimates can thus be derived as follows:

```
burnin=6000
thin = 100
end=1e04
x<-seq(burnin+1,end,thin)

paras<-rbind(paras1[x,], paras2[x,])

#posterior estimates for alpha
round(quantile(paras$alpha, probs=c(0.5, 0.025, 0.975)), 4)
#>   50%  2.5% 97.5%
#> 2e-04 0e+00 6e-04

#posterior estimates for beta
round(quantile(paras$beta, probs=c(0.5, 0.025, 0.975)), 3)
#>   50%  2.5% 97.5%
#> 0.020 0.001 0.047

#all of the medians can be quickly extracted with the apply function
apply(paras,2,median)
#>         sample log_likelihood           corr          alpha           beta
#>   7.961000e+03  -1.132075e+04   7.200000e+01   1.825450e-04   2.024490e-02
#>         lat_mu         lat_sd              c              d            k_1
#>   7.389335e+00   5.510015e+00   2.072260e+01   3.742715e+00   1.716485e+00
#>           mu_1           mu_2          p_ber       phi_inf1       phi_inf2
#>   2.138570e-05   2.384495e-06   9.807640e-02   3.417495e+00   1.778560e+00
#>      rho_susc1      rho_susc2         nu_inf       tau_susc         beta_m
#>   4.566105e-01   1.467015e+00   1.521275e-01   9.261845e-02   5.000000e-01
```

Producting density plots for a parameter of interest:

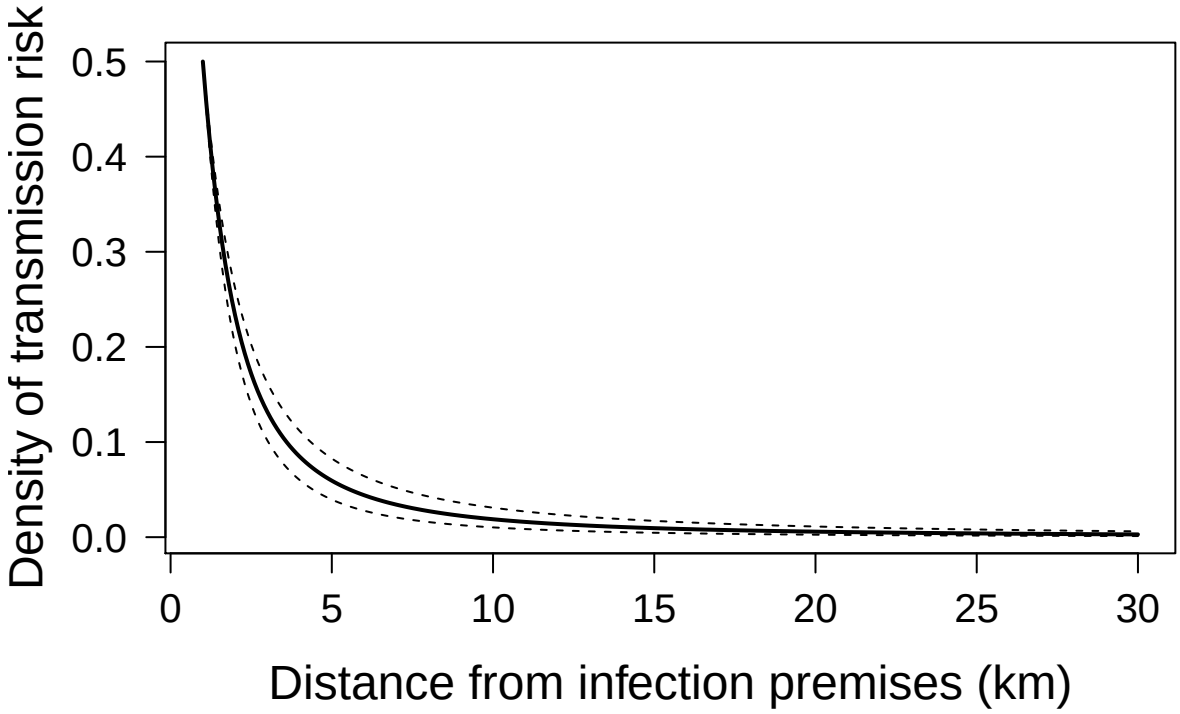
```
plot(density(paras1$alpha[x]), col='red', cex.main=2,
  main=expression(paste(plain("Density of "),alpha,plain(", by run"))))
lines(density(paras2$alpha[x]), col='blue')
```



The inference for the `k_1` parameter, the scaling parameter of the spatial kernel, can be used to plot the estimated spatial risk around each infected premises, as follows:

```
#posterior estimates for k_1 (spatial kernel scaling)
k1<-quantile(paras$k_1, probs=c(.5, .025, .975))
round(k1, 3)
#>   50%  2.5% 97.5%
#> 1.716 1.495 1.983

#x-axis range
x<-seq(1,30,0.1)

#median estimates
yq50 = 1 / (1 + x^k1[1])
#lower and upper 95% credible intervals
yl = 1 / (1 + x^k1[2])
yu = 1 / (1 + x^k1[3])

plot(x, yq50, type='n', lwd=2, las=1,
```

```
     xlab='Distance from infection premises (km)',
     ylab ='Density of transmission risk',
     cex.lab=1.5, cex.axis=1.25)
     lines(x, yl, lty=2)
     lines(x, yu, lty=2)
     lines(x, yq50, lwd=2)
```



Inferred infected sources

Pre-prepared infected source data for each chain have been inputted by reading in the comma separated `infected_source_current.csv` files produced by independently seeded `infer()` runs on the same input data.

Let's inspect the `head` and `tail` of each of these datasets:

```
#load pre-prepared infer output data
inf1<-read.csv("run1_infected_source_current.csv", header=F)
inf2<-read.csv("run2_infected_source_current.csv", header=F)

#rename the columns
colnames(inf1)<-colnames(inf2)<-cov$k

#inspect the top and bottom for chain 1
head(inf1[,1:10])
#>      0 1  2 3 4    5    6  7 8  9
#> 1 9999 4 81 0 0 9999 9999 20 4 28
#> 2 9999 4 81 0 0 9999 9999 20 4 28
#> 3 9999 4 81 0 0 9999 9999 20 4 28
#> 4 9999 4 81 0 0 9999 9999 20 4 28
#> 5 9999 4 81 0 0 9999 9999 20 4 28
#> 6 9999 4 81 0 0 9999 9999 20 4 28
tail(inf1[,1:10])
#>          0 1  2 3 4 5 6  7 8 9
#> 10005 9999 5 71 0 0 0 0 97 0 0
#> 10006 9999 5 71 0 0 0 0 97 0 0
#> 10007 9999 5 71 0 0 0 0 97 0 0
#> 10008 9999 5 71 0 0 0 0 97 0 0
#> 10009 9999 5 71 0 0 0 0 97 0 0
#> 10010 9999 5 71 0 0 0 0 97 0 0
```

Using the same burn-in as previously decided:

```
burnin
#> [1] 6000
thin
#> [1] 100
end
#> [1] 10000
x<-seq(burnin+1,end,thin)
n<-nrow(cov)

#create lists that store for each individual:
#- a frequence table of iterations featuring each infected source (inf.l)
#- the source with the highest modal posterior support (inf.mode.l)
#- the posterior support for each infected source (inf.support.l)
inf.l<-list()
inf.mode.l<-numeric()
inf.support.l<-numeric()
for(i in 1:n){
  inf.l[[i]]<-table(c(inf1[x,i], inf2[x,i]))
  inf.mode.l[i]<-as.numeric(attr(which.max(inf.l[[i]]), 'names'))
  inf.support.l[i]<-
    as.numeric(inf.l[[i]][which.max(inf.l[[i]])]/sum(inf.l[[i]]))
}

#inspect these outputs for individual k=3
inf.l[[4]]
#>
#>  0  5  8
#> 77  2  1
inf.mode.l[4]
#> [1] 0
inf.support.l[4]
#> [1] 0.9625
```

The highest modal posterior supports can be combined into an edgelist:

```
el<-data.frame(to=1:n, from=inf.mode.l)

head(el)
#>   to from
#> 1  1 9999
#> 2  2    5
#> 3  3   71
#> 4  4    0
#> 5  5    0
#> 6  6    0
```

And can be compared to known or suspected sources:

```
#which are correctly inferred for the IPs only,
#we ignore the sources for the non-IPs
acc <- ifelse(accTable[ips] == el$from[ips], 1, 0)

#number correct
sum(acc)
#> [1] 71

#% accuracy of sources
round(sum(acc)/length(ips),2)
#> [1] 0.86
```

This accuracy could be better if this inference were run for at least 100,000 MCMC iterations. These example data are based on only 10,000 iterations.

The accuracy of sources considered with consensus support or better can be estimated as follows:

```
#how many have greater than consensus support?
nsup<-length(which(inf.support.l[ips]>0.5))
nsup
#> [1] 70

#proportion of individuals with an inferred source with consensus support
round(nsup/length(ips),2)
#> [1] 0.84

#accuracy for the inferred sources of these individuals only
round(sum(acc[inf.support.l[ips]>0.5])/nsup, 2)
#> [1] 0.93
```

Inferred key timings

Pre-prepared data on the inferred timing of exposured of each individual for each iteration in each chain have been inputted by reading in the comma separated `t_e_current.csv` files produced by independantly seeded `infer()` runs on the same input data.

```
te1<-read.csv("run1_t_e_current.csv", header=F)
te2<-read.csv("run2_t_e_current.csv", header=F)
colnames(te1)<-colnames(te2)<-cov$k

round(head(te1[,1:10]), 3)
#>       0      1     2      3      4     5      6      7      8    9
#> 1 0.425 12.135 60.91 13.032 11.244 2.128 14.355 54.335 12.429 29.4
#> 2 0.425 11.887 60.91 13.032 11.244 2.128 14.355 54.335 12.429 29.4
#> 3 0.425 11.887 60.91 13.032 11.244 2.128 14.355 54.335 12.429 29.4
#> 4 0.425 11.887 60.91 13.032 11.244 2.128 14.355 54.335 12.429 29.4
#> 5 1.943 11.887 60.91 13.032 11.244 2.128 14.355 54.335 12.429 29.4
#> 6 1.943 11.887 60.91 13.285 11.244 2.128 14.355 54.335 12.429 29.4
```

Again we'll use the same burn-in parameters. This time collating lists with posterior median estimates of the day of exposure for each individual with 95% Bayesian credible intervals.

```
burnin
#> [1] 6000
thin
#> [1] 100
end
#> [1] 10000
x<-seq(burnin+1,end,thin)

te.l<-list()
te.q.l<-list()
for(i in 1:n){
  te.l[[i]]<-c(te1[x,i], te2[x,i])
  te.q.l[[i]]<-quantile(te.l[[i]], probs=c(0.5, 0.025, 0.975))
}

#inspecting some of the inferred exposure timings
round(te.q.l[[1]], 2)
#>   50%  2.5% 97.5%
#>  2.45  0.16  5.10
round(te.q.l[[2]], 2)
#>   50%  2.5% 97.5%
#> 12.11  8.95 15.21
round(te.q.l[[3]], 2)
#>   50%  2.5% 97.5%
#> 88.86 84.73 94.94
```

Pre-prepared data on the inferred timing of onset of infectiousness of each individual for each iteration in each chain have been inputted by reading in the comma separated `t_i_current.csv` files produced by independantly seeded `infer()` runs on the same input data.

```
ti1<-read.csv("run1_t_i_current.csv", header=F)
ti2<-read.csv("run2_t_i_current.csv", header=F)
colnames(ti1)<-colnames(ti2)<-cov$k
```

Again we'll use the same burn-in parameters. This time collating lists with posterior median estimates of the day of onset of infectousness for each individual with 95% Bayesian credible intervals.

```
burnin
#> [1] 6000
thin
#> [1] 100
end
#> [1] 10000
x<-seq(burnin+1,end,thin)

ti.l<-list()
ti.q.l<-list()
for(i in 1:n){
  ti.l[[i]]<-c(ti1[x,i], ti2[x,i])
  ti.q.l[[i]]<-quantile(ti.l[[i]], probs=c(0.5, 0.025, 0.975))
}

#inspecting some of the inferred exposure timings
round(ti.q.l[[1]], 2)
#>   50%  2.5% 97.5%
#>  5.78  4.04  6.88
round(ti.q.l[[2]], 2)
#>   50%  2.5% 97.5%
#> 14.84 10.30 18.72
round(ti.q.l[[3]], 2)
#>   50%  2.5% 97.5%
#> 92.43 90.20 96.46
```

Inferred sequences

A sequence is written out every nth iteration, as set with the parameter `pars.aux$n_output_gm`. In these iterations, whenever an individual is infected, sampled or when it infects another individual, a sequence is written. So each individual can have a different number of sequences recorded from other individuals, even from itself in different iterations.

Each `pars.aux$n_output_gm` iteration, the timing of recorded sequences is captured in the output file `seqs_t_current_csv` with 1 row per individual, comma-separated.

```
# read in the timings data from the first chain
nt.t1<-readLines('run1_seqs_t_current.csv')

#read in the sequence data
#this is a large file and takes several seconds to load
nt.seq1<-read.csv('run1_seqs_current.csv', header=F)

#check the dimensions of the sequence data
dim(nt.seq1)
#> [1] 2840 7667

#convert the sequences to their nucleotide base letters
nt.seq1[nt.seq1==1]<-"a"
nt.seq1[nt.seq1==2]<-"g"
nt.seq1[nt.seq1==3]<-"t"
nt.seq1[nt.seq1==4]<-"c"

library(BORIS)
#>
#> Attaching package: 'BORIS'
#> The following objects are masked _by_ '.GlobalEnv':
#>
#>     inf1, inf2, nt.seq1, nt.t1, paras1, paras2, te1, te2, ti1, ti2
#use seqlookup function to show timings of sequences recorded for individual with index `k` in iteratio
seqlookup(k=1, it=2,seq.ts=nt.t1, accTable=accTable)$ts
#> [1] 11.9979 18.7289 20.2577

#show the first 10 nucleotides of the sequence data
#for the corresponding sequences for individual `k`
#in iteration `it`
nt.seq1[seqlookup(k=1, it=2,seq.ts=nt.t1,
                  accTable=accTable)$seq.ts,1:10]
#>     V1 V2 V3 V4 V5 V6 V7 V8 V9 V10
#> 264  g  a  t  a  g  a  t  c  t   c
#> 265  g  a  t  a  g  a  t  c  t   c
#> 266  g  a  t  a  g  a  t  c  t   c
```

So, for the second individual (`k=1`) in the second iteration, there are 3 sequences.

Outbreak simulation with BORIS

In this example, we will:

1. Prepare the inputs for simulation
2. Simulate an outbreak with corresponding genomic data

Scenario (continued)

We have inferred the transmission network between the 100 known infected farms and key parameters (in the earlier exercise).

If we had detected this outbreak on day 14 into the outbreak, we could run the reconstruction as previously and forward simulate until day 100 using the posterior estimates of the key parameters.

This simulation follows the same formulation as the inference in BORIS. It does not incorporate control actions. More complex simulators like the Australian Animal Disease Spread model (Bradhurst, et al., 2015) should be considered for testing control strategies, especially across larger scales.

Preparing inputs for simulation

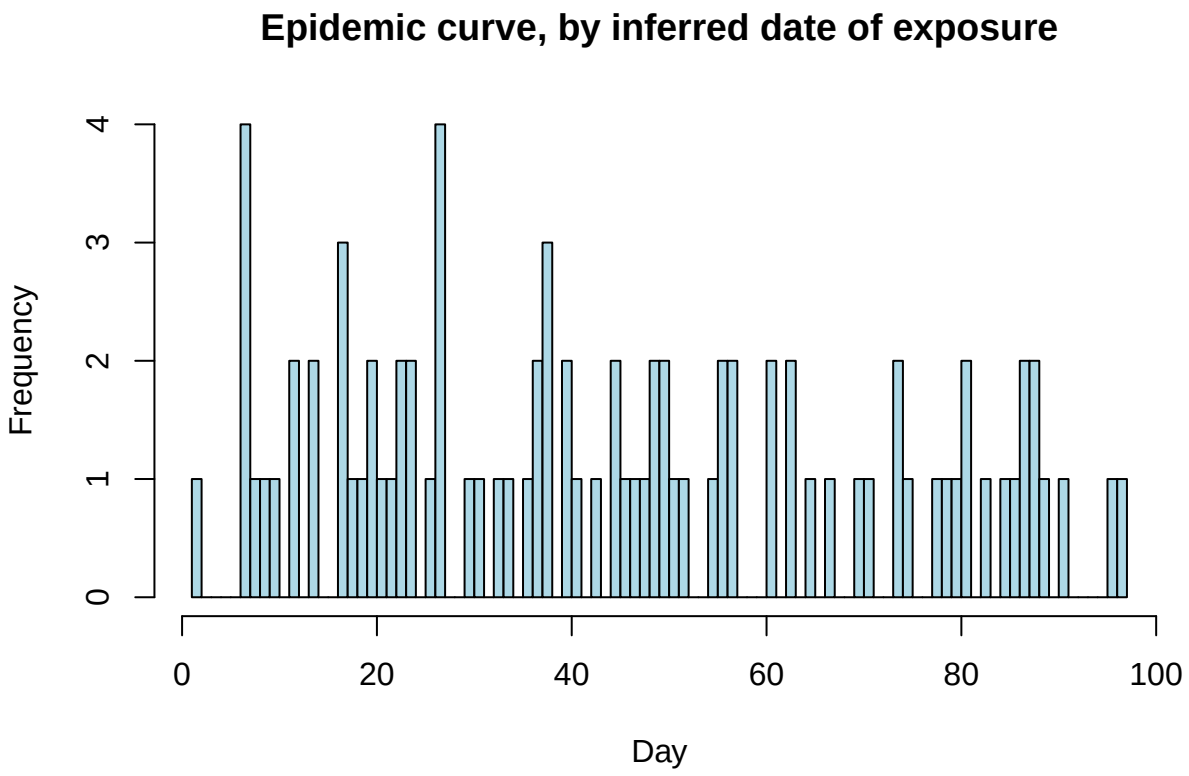
The `covariate` data needs to be inputed as a dataframe as specified in the help for `data(sim.epi.input)`.

```
library(BORIS)

#structure of the data that we are aiming for forwards simulation
data(sim.epi.input)
head(sim.epi.input)
#>   k  coor_x   coor_y   t_e   t_i   t_r ftype0 ftype1 ftype2 herdn status
#> 1 0 144.901 -36.4348 0e+00 9e+06 9e+06      0      1      0  3209      2
#> 2 1 144.924 -36.4288 9e+06 9e+06 9e+06      0      0      1  1705      1
#> 3 2 144.924 -36.4288 9e+06 9e+06 9e+06      1      0      0   133      1
#> 4 3 144.903 -36.4205 9e+06 9e+06 9e+06      1      0      0   213      1
#> 5 4 144.907 -36.4179 9e+06 9e+06 9e+06      1      0      0   127      1
#> 6 5 144.898 -36.4310 9e+06 9e+06 9e+06      0      0      1     6      1

#we'll start with the covariate data from the earlier inference
epi.sim<-cov
head(epi.sim)
#>   k   coor_x    coor_y t_e t_i t_r ftype herdn initial_source
#> 1 0 146.0816 -38.32329   1   3  29     1  3209           9999
#> 2 1 146.0831 -38.32280  10  12  27     2  1705              0
#> 3 2 146.0831 -38.32280  95  97 100     0   133             88
#> 4 3 146.0817 -38.32212  12  14  35     0   213              4
#> 5 4 146.0820 -38.32190   9  11  37     0   127             65
#> 6 5 146.0814 -38.32298   7   9  26     2     6              0

#and use the posterior median estimates from the inference
#for the timing of exposure and onset of infectiousness
epi.sim$t_e<-sapply(te.l, median)
```

```
epi.sim$t_i<-sapply(ti.l, median)

#re-order the data by day of exposure and re-index
epi.sim<-epi.sim[order(epi.sim$t_e),]
epi.sim$k<-1:n-1

#inspect the data
head(epi.sim)
#>    k   coor_x    coor_y      t_e      t_i t_r ftype herdn initial_source
#> 1  0 146.0816 -38.32329 2.446945  5.77745  29     1  3209           9999
#> 13 1 146.0561 -38.33806 6.973345 13.65830  39     0   145             70
#> 6  2 146.0814 -38.32298 7.120900  9.50372  26     2     6              0
#> 66 3 146.1583 -38.31171 7.247030 10.58220  34     2   661              5
#> 4  4 146.0817 -38.32212 7.403495 15.03070  35     0   213              4
#> 5  5 146.0820 -38.32190 7.805330 14.88170  37     0   127             65
tail(epi.sim)
#>       k  coor_x   coor_y   t_e   t_i t_r ftype herdn initial_source
#> 144 144 145.782 -38.3157 9e+06 9e+06 100     0   244           9999
#> 145 145 145.877 -38.3403 9e+06 9e+06 100     2     5           9999
#> 147 146 146.192 -38.3988 9e+06 9e+06 100     0   308           9999
#> 148 147 145.916 -38.2808 9e+06 9e+06 100     0    92           9999
#> 149 148 145.872 -38.2744 9e+06 9e+06 100     1     5           9999
#> 150 149 146.242 -38.4174 9e+06 9e+06 100     0     6           9999
```

To display an epidemic curve at this point:

```
ips<-which(epi.sim$t_e<pars.aux$t_max)
tmp<-round(epi.sim$t_e[ips],0)

hist(tmp, breaks = 1:max(tmp), col='lightblue',
     main='Epidemic curve, by inferred date of exposure', xlab='Day')
```



Next, we'll build the final dataset for simulation:

```
epi.sim.final<-data.frame(k=epi.sim$k,
                      coor_x=epi.sim$coor_x,
                      coor_y=epi.sim$coor_y,
                      t_e=epi.sim$t_e,
                      t_i=epi.sim$t_i,
                      t_r=epi.sim$t_r,
                      ftype0 = (epi.sim$ftype==0)*1,
                      ftype1 = (epi.sim$ftype==1)*1,
                      ftype2 = (epi.sim$ftype==2)*1,
                      herdn = epi.sim$herdn,
                      status = 1)
```

The `status` variable denotes the highest level for each farm: 1 == Susceptibles throughout the time period from t0 to t_max, 2 == Exposed at some stage between t0 to t_max, 3 == Infectious at some stage, 4 == Recovered/Removed.

To simulate from day 14 onwards:

```
t0 <- 14
ips<-which(epi.sim.final$t_e< t0)
length(ips)
#> [1] 11
#there are 11 IPs exposed by this point
```

```
#their inferred timing of exposure
round(epi.sim.final$t_e[ips],1)
#>  [1]  2.4  7.0  7.1  7.2  7.4  7.8  9.2 10.0 12.1 12.2 13.9

#set the status
epi.sim.final$status[epi.sim.final$t_e < t0]<-2
epi.sim.final$status[epi.sim.final$t_i < t0]<-3
epi.sim.final$status[epi.sim.final$t_r < t0]<-4
table(epi.sim.final$status)
#>
#>   1   2   3
#> 139   7   4

#reset any events that haven't occurred yet to unassigned
epi.sim.final$t_e[epi.sim.final$status==1] <- 9e6
epi.sim.final$t_i[epi.sim.final$status==1] <- 9e6
epi.sim.final$t_r[epi.sim.final$status==1] <- 9e6

epi.sim.final$t_i[epi.sim.final$status==2] <- 9e6
epi.sim.final$t_r[epi.sim.final$status==2] <- 9e6

epi.sim.final$t_r[epi.sim.final$status==3] <- 9e6

#inspect the finished product
head(epi.sim.final)
#>   k   coor_x    coor_y      t_e          t_i   t_r ftype0 ftype1 ftype2
#> 1 0 146.0816 -38.32329 2.446945 5.77745e+00 9e+06      0      1      0
#> 2 1 146.0561 -38.33806 6.973345 1.36583e+01 9e+06      1      0      0
#> 3 2 146.0814 -38.32298 7.120900 9.50372e+00 9e+06      0      0      1
#> 4 3 146.1583 -38.31171 7.247030 1.05822e+01 9e+06      0      0      1
#> 5 4 146.0817 -38.32212 7.403495 9.00000e+06 9e+06      1      0      0
#> 6 5 146.0820 -38.32190 7.805330 9.00000e+06 9e+06      1      0      0
#>   herdn status
#> 1  3209      3
#> 2   145      3
#> 3     6      3
#> 4   661      3
#> 5   213      2
#> 6   127      2
```

Parameterising the simulation

We will parameterise this simulation with posterior median estimates from our earlier inference run. In practice, multiple simulations would be undertaken, rather than just one. If each were parameterised with a set of randomly drawn sample from the posterior distribution, i.e., all parameters where from one iteration, once the model had converged, then the simulations may be considered a posterior sample themselves. Running a couple of hundred such multidimensional samples from the posterior distribution as simulations can be a means to estimating credible intervals and quantifying uncertainty in the predictions.

All of the parameters can be taken directly from the posterior inference into the simulation, except for those representing the hyper-parameters of the Gamma distribution for the latent period (`a` and `b`). The formulation of outputs in the inference code was for the mean and SD of such a Gamma distribution (`lat_mu` and `lat_sd`, respectively). It is straightforwards to convert these to the desired shape parameters using the

following formulae:

$shape = mean \times mean \div var$

$mean = shape \times scale$

$var = shape \times scale \times scale$

```
a <- median(paras$lat_mu)^2 / (median(paras$lat_sd)^2)
b <- median(paras$lat_mu)/a
```

para.key.sim

The key simulation parameters are setup as follows based on the earlier posterior median estimates. For details see the help for `sim.param.key`.

```
para.key.sim <- data.frame('alpha' = median(paras$alpha),
                           'beta' =  median(paras$beta),
                           'mu_1' = median(paras$mu_1),
                           'mu_2' = median(paras$mu_2),
                           'a' = a,
                           'b' = b,
                           'c' = median(paras$c),
                           'd' = median(paras$d),
                           'k_1' = median(paras$k_1),
                           'p_ber' = median(paras$p_ber),
                           'phi_inf1' = median(paras$phi_inf1),
                           'phi_inf2' = median(paras$phi_inf2),
                           'rho_susc1' = median(paras$rho_susc1),
                           'rho_susc2' =median(paras$rho_susc2),
                           'nu_inf' = median(paras$nu_inf),
                           'tau_susc'= median(paras$tau_susc),
                           'beta_m'= median(paras$beta_m))
```

pars.aux.sim

The auxillary parameters for controlling the simulation are setup as follows. For details see the help for `sim.param.aux`.

```
pars.aux.sim <- data.frame('n' = nrow(epi.sim.final),
                           'seed' = 2467,
                           'n_base' = 7667,
                           'n_seq' = 5,
                           't_max' = 100 - t0,
                           'unassigned_time' =  9e+6,
                           'sample_range' = 10,
                           'partial_seq_out' = 0,
                           'n_base_part' = 1000,
                           'n_index' = 1,
                           'coord_type' = 'longlat',
                           'kernel_type' = 'power_law',
```

```
                            'latent_type' = 'gamma',
                            'opt_k80' = 1,
                            'opt_betaij' = 1,
                            'opt_mov' = 0,
                            'n_mov' = 60,
                            stringsAsFactors = F)
```

Animal movement data

An **animal movement/contact-tracing dataset** is required to be inputted as a dataframe as specified in the help for `data(sim.moves.input)`, irrespective of whether it is used (i.e. whether the option `parsAux$opt_mov` $= 1$ or $= 0$. If `parsAux$opt_mov` is set to 0, you can just use the example dataset `data(sim.moves.input)`.

```
data(sim.moves.input)

head(sim.moves.input)
#>   from_k to_k t_m
#> 1      0    1   9
#> 2      0    2   9
#> 3      0    3   9
#> 4      0    4   9
#> 5      0    5   9
#> 6      0    6   9
```

Simulate an outbreak with corresponding genomic data

Then you are ready to run the simulation as follows:

```
sim.out<-sim(epi.inputs = epi.sim.final,
             moves.inputs = sim.moves.input,
             parsKey = para.key.sim,
             parsAux = pars.aux.sim,
             inputPath = "./inputs",
             outputPath = "./outputs")
#> Input path: ./inputs\
#> Output path: ./outputs\
#>
#> Outbreak simulated successfully.
#>
```

This function:

1. Creates input and output directories at the locations specified, or deletes everything existing in those directories.
2. Checks the ranges of key inputs to see that they are within reasonable limits.
3. Writes a set of input files to the `inputPath`.
4. Runs the simulation storing key outputs into an object `sim.out` and also writing output files to the `outputPath`.

Inspecting simulation outputs

The epidemiological dataset with simulated values for timing of exposure, onset and end of the infectious period for each individual.

```
sim.out$epi.sim[1:6,]
```

```
#>      k   coor_x    coor_y t_e  t_i  t_r ftype0 ftype1 ftype2 herdn
#> [1,] 0 146.0816 -38.32329 2.4  5.8 15.0      0      1      0  3209
#> [2,] 1 146.0561 -38.33806 7.0 13.7 20.9      1      0      0   145
#> [3,] 2 146.0814 -38.32298 7.1  9.5 36.0      0      0      1     6
#> [4,] 3 146.1583 -38.31171 7.2 10.6 26.7      0      0      1   661
#> [5,] 4 146.0817 -38.32212 7.4 10.7 24.7      1      0      0   213
#> [6,] 5 146.0820 -38.32190 7.8 12.5 22.7      1      0      0   127
```

The unique ordered identifier (i.e. k) of the simulated infected source of each individual. Those with a simulated source = 9999 have been infected from an external source.

```
head(sim.out$infected_source, 20)
#>  [1] 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999  -99  -99   40
#> [15]    7   17  109   98   16   66
```

Any with a simulated source = -99 have not been infected in this simulation.

```
tmp<-which(sim.out$infected_source == -99)
length(tmp)
#> [1] 59
```

The number and percentage of individuals in the population that were infected can be calculated as:

```
inf <- which(sim.out$infected_source != -99)
length(inf)
#> [1] 91

length(inf)/length(sim.out$infected_source)
#> [1] 0.6066667
```

The simulated timing of sampling per individual. Those with the `unassigned_time` of 9e6 have not been sampled.

```
head(floor(sim.out$t_sample), 20)
#>  [1]      12       9       7       8       9      10      11      12
#>  [9]      13      15      14 9000000 9000000      39      29      56
#> [17]      39      50      53      78
```

Simulated sequence data

The sequence data itself can be found in the files `subject_x_nt.txt` with the `x` repesenting the identified (`k`) for each individual. The nucleotides bases are represented as 1=A, 2=G, 3=T, 4=C. If an individual has a sequence simulated at more than one time-point (i.e. when it was exposed, then subsequently at the time when it infected another individual(s), and the possibly again if sampled) then each simulated sequence is written to a new line, comma-separated. The times corresponding to these sequences are stored in the files `subject_x_t_nt.txt`.

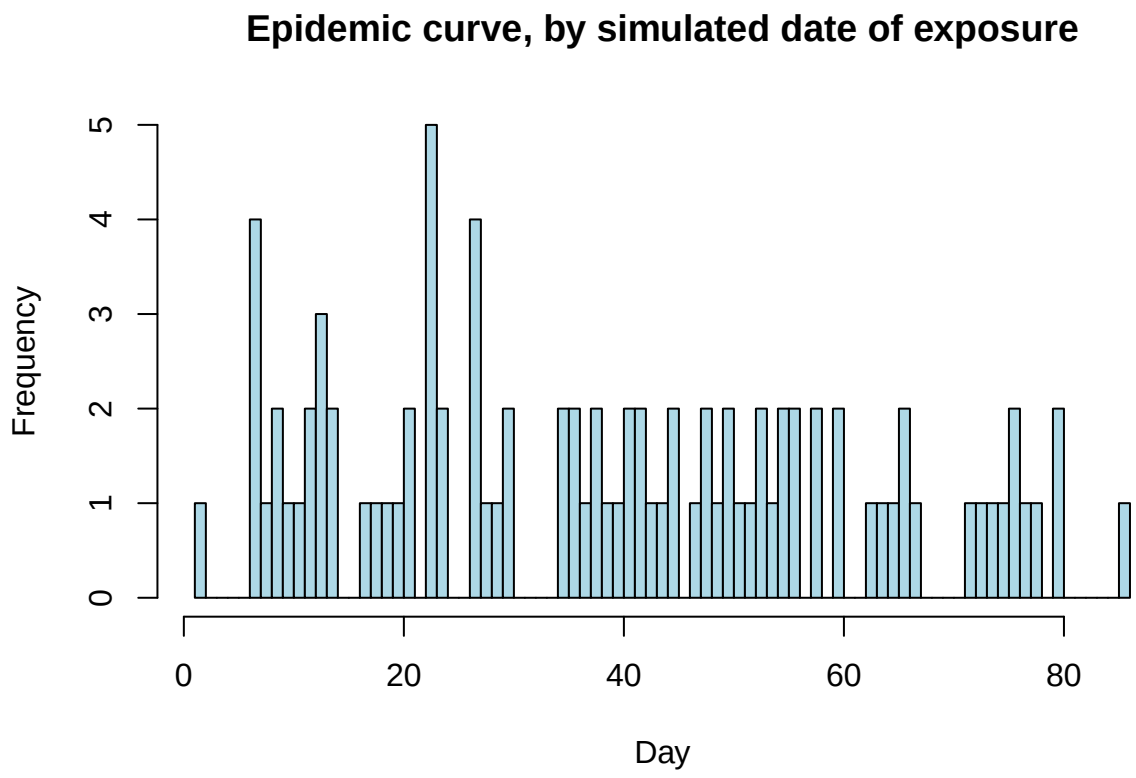
To write a simulated set of sequences for an individual out to FASTA using the `ape` library:

```
library(ape)

#read in the sequences from individual k=0
fn <- paste0(sim.out$outputPath, "subject_0_nt.txt")
seqs1 <- as.matrix(read.csv(fn, header=F))

#the sequences for this individul are stored in 19 rows
# nucleotides are coded as 1=A, 2=G, 3=T, 4=C
dim(seqs1)
#> [1]    3 7667

#convert the sequences to their nucleotide base letters
seqs1[seqs1==1]<-"a"
seqs1[seqs1==2]<-"g"
seqs1[seqs1==3]<-"t"
seqs1[seqs1==4]<-"c"

#name the rows, which become the tip labels in the fasta file.
#for this we will import the timings associated with each sequence
fn <- paste0(sim.out$outputPath, "subject_0_t_nt.txt")
seqs1_t <- as.numeric(unlist(read.table(fn)))

rownames(seqs1) <- paste0("k0_", round(seqs1_t,3))

#write these out to a fasta file
library(ape)
write.dna(seqs1, file='seqs1.fasta', format = 'fasta', nbcol = -1)

#inspect snps in these data
seq.tab <- apply(seqs1, MARGIN = 2, table)
snp.v<-sapply(seq.tab, length)
snps<-as.numeric(which(snp.v>1))
seqs1[1:3,snps]
#>           V1129 V2141 V2184 V2527 V3218 V5157 V5171 V5594 V6448
#> k0_2.447  "g"   "a"   "t"   "a"   "a"   "c"   "c"   "g"   "t"
#> k0_11.486 "a"   "g"   "t"   "g"   "a"   "t"   "t"   "a"   "c"
#> k0_12.365 "a"   "g"   "c"   "g"   "g"   "t"   "t"   "a"   "c"
```

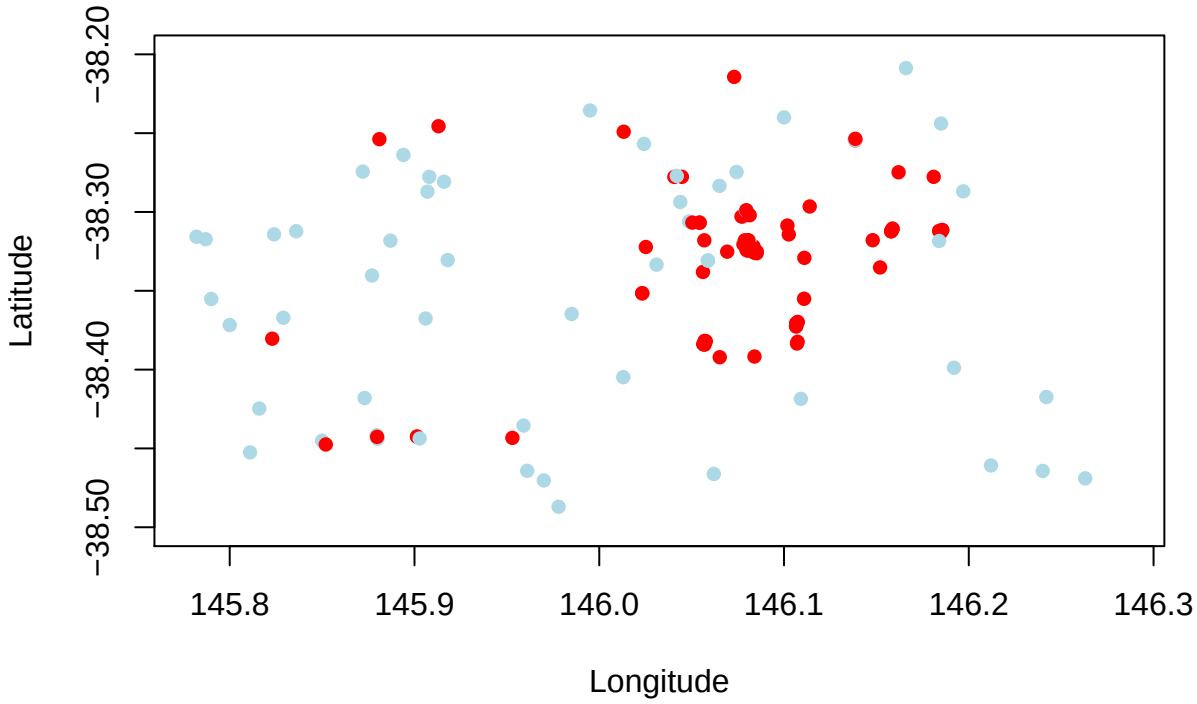
Simulated epicurve and map

Finally plot an epicurve and map of the simulated outbreak:

```r
#display an epidemic curve
epi.sim.out<-as.data.frame(sim.out$epi.sim)
ips<-which(epi.sim.out$t_e < pars.aux$t_max)
tmp<-round(epi.sim.out$t_e[ips],0)

hist(tmp, breaks = 1:max(tmp), col='lightblue',
     main='Epidemic curve, by simulated date of exposure', xlab='Day')
```




```r
#plot a map
col<-rep('lightblue',n)
col[ips]<-'red'
plot(x = epi.sim.out$coor_x, y = epi.sim.out$coor_y,
     axes = TRUE, xlab = "Longitude", ylab = "Latitude",
     xlim=c(145.7795,146.2856), ylim=c(-38.5,-38.2),
     pch = 16, col = col, cex=1)
```


Latitude
−38.20
−38.30
−38.40
−38.50
145.8
145.9
146.0
146.1
146.2
146.3
Longitude