Visualising an infectious disease outbreak

OJ Watson

2017-10-08

Contents

Objectives:

Last week you participated in an outbreak of a suspected novel virus 'DIDE-DISEASE', and you have just now analysed this data using excel. Excel is a very useful tool for presenting the data and carrying out some analysis to estimate the epidemiological parameters. However, we can also carry out the same analysis using the programming language R, while also extending our analysis to include additional data visualization. Data visualisation can be a very useful tool for revealing important features of outbreak dynamics, and hopefully by the end of this practical you will see that it is more than just plotting graphs with x and y axes.

In this practical you will use an R package called *outbreakteachR* that was built to analyse the data you have collected within your excel sheet. An R package is a collection of R functions, data and code designed to carry out a group of desired tasks. *outbreakteachR* includes functionality to animate the outbreak and visualise the network of transmission events within the outbreak. It also contains functionality to simulate outbreaks using a mathematical model, which will be used to provide an example of how models can be useful tools for studying outbreaks.

Getting started:

We will begin with a brief introduction into R, before exploring *outbreakteachR*. This practical will be carried out entirely in R, so begin by opening R studio. R studio is a program that provides facilities that make writing R code easier, and the desktop version can be downloaded from *here*. Once open you will be presented with a window that looks similar to that shown in Figure 1.

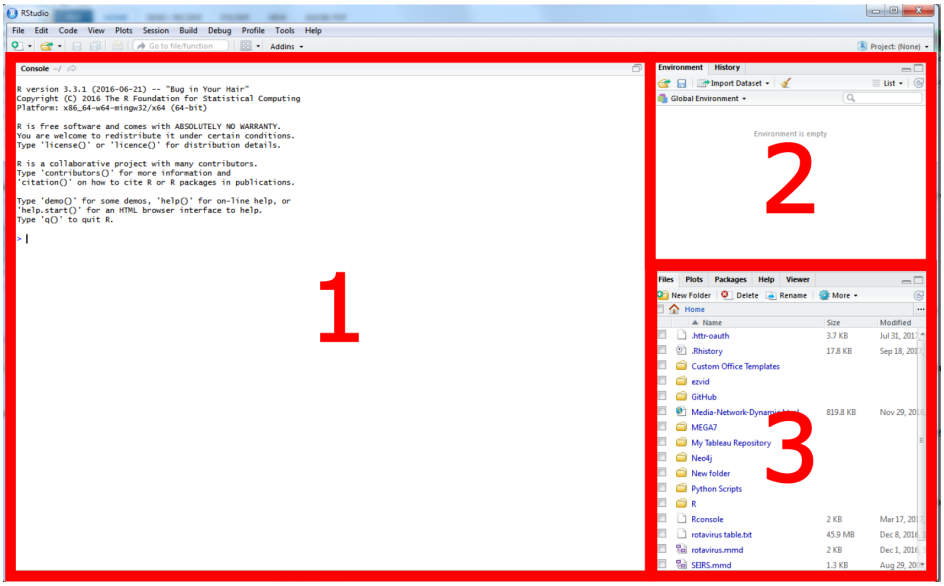


Figure 1: R Studio Window

In this window we can see the (1) console, (2) environment viewer and (3) file explorer. The console is where we can run code and see the output of that code. The environment viewer shows us what variables are currently stored in memory within our current R session. If we click on the history tab within this window we can also see our R history, which is where our previously executed R code is shown. Lastly the file explorer is similar to a normal explorer window and by default should show the location of where our current working directory is. The other tabs in this window can also show us any plots we have created, information about the packages we have loaded and a help window.

Today we will want to write a few lines of code that will install *outbreakteachR* and carry out some visualisation. We can write R directly into the console that will be run when you hit Enter afterwards. For example, the following will create a variable called *a* that will be set equal to 2:

```
a <- 2
a
```

```
## [1] 2
```

We can check that *a* is in fact equal to 2 by looking into the environment viewer and seeing that there is a now a value for *a* showing 2. We can also write lines of code within a script. To start a new script go to *File > New File > R Script*. This will open a new panel within our window in the top left, pushing the console down.

In this file we can now write consecutive lines of R code. For example we can write functions that will carry out consecutive calculations and manipulate R variables and numbers that are passed to them as arguments. For example, `sqrt(4)` is a function that will return the square root of the argument passed to it, which in this case is 4, and will thus return 2. `sqrt()` is a function included in base R, however, we can also write functions ourselves. For example, a function to convert Fahrenheit to Celsius:

```
# function to convert the provided Fahrenheit argument to Celsius
fahr_to_celsius <- function(fahrenheit){
celsius <- (fahrenheit - 32) * (5/9)
return(celsius)
}
```

Write this within our new R script window, select all the lines the function uses and press *Ctrl+Enter* to run the function. We will see that the function now appears within our environment window under a subheading **Functions**. We can now use that function to convert a Fahrenheit temperature without having to write out the function each time:

```
fahr_to_celsius(fahrenheit = 275)
```

```
## [1] 135
```

The above has been a quick overview of how to start R studio and use it to write a function. Earlier we discussed how an R package is simply a collection of R functions, data and code designed to carry out a group of desired tasks. *outbreakteachR* is no different, and is a collection of functions designed to read in the outbreak dataset from the Excel file, and carry out a series of visualisations and analysis, which we will cover now.

outbreakteachR

The *outbreakteachR* R package has been designed as an additional tool in helping assist with running the "paper outbreak" practical. This includes importing the data from the excel sheet you have created, plotting and animating the infection dynamics and carrying out some simple simulations of a SIR mathematical model.

Installing *outbreakteachR*

In order to use the *outbreakteachR* package we must first install it. To do this we need to install it from the github repository where the package is maintained, which will require us to first install a package called *devtools*.

```
# First install devtools using the install.packages() function
install.packages(pkgs = "devtools", repos = "https://cloud.r-project.org/")

# Second install the package using the install_github() function
devtools::install_github("mrc-ide/outbreakteachR",ref="mrc-branch")
```

The above code will first install the package *devtools*, which is then used to install the *outbreakteachR* package via the `devtools::install_github()` function. the `::` that were used after devtools allows us to use functions within the package without loading the package using the `library()` function. This is often a useful way to use functions from within a package as it makes it easier for someone who is reading your code to know where a function has come from. However, to save time during this practical we will use the `library()` function to load the *outbreakteachR* package into

our R session.

```
# Load package - Warning messages will appear however these are not a problem
library(outbreakteachR)
```

To see that this has happened, you can click on *Global Environment* within the Environment window and be able to see *outbreakteachR* listed along with several other packages. Functions that are made available within packages that are within our environment can be used without having to use the `::` as we will see now.

Loading the data

We will now read in the data from our saved outbreak dataset .xlsx files. In order to use your own dataset the package assumes that the data has been collected and stored in a similar fashion to that presented within your practical. As such it will import the main data from the first sheet in the excel document between columns A to O inclusive.

To view an example of how the data should be laid out in the excel sheet, please run the following code and have a look at sheet 1:

```
# Open the excel sheet within excel
system(paste0("open ",
              "\"",
              system.file("extdata/2016_solutions_final.xlsx",
                          package="outbreakteachR"),
              "\"")
       )
```

To read in your dataset we will use the function `outbreak_dataset_read`, which accepts as the first argument the file path to where your excel file is saved. If you are using your own collected data, change the xlsx.file argument to the file path where your dataset is saved.

```
# Remember to replace the argument for xlsx.file
# to the local file path where your saved worksheet is
# If it is on your desktop it will look something
# like xlsx.file = "C:/Users/Bob/Desktop/savedworksheet.xlsx"

outbreak.dataset <- outbreak_dataset_read(xlsx.file = "C:/Users/Oliver/data.xlsx")
```

If you do not have a dataset and wish to use the 2016 dataset, then you can do this by using the following code.In the reaminder of the tutorial we will be using the 2016 dataset as an example (the one that we opened earlier).

```
# If you don't have a working dataset, but want to be able to carry out the
# following analysis then you can create the dataset by running the following:

outbreak.dataset <- outbreak_dataset_read(
```

```
    xlsx.file = system.file("extdata",
                            "2016_solutions_final.xlsx",
                            package="outbreakteachR"))
```

The `outbreak_dataset_read` function may throw warnings highlighting rows where there may be errors in the excel file. If this is the case, review the excel file and correct any rows that are potentially missing data or have negative values for the generation time etc. The function will also attempt some basic data imputation to work out some missing data.

To view the format that our dataset has been imported as we will use the `str` function that displays the structure of an R object, and is very useful to quickly glance at an object.

```
# View the dataset columns
str(outbreak.dataset,width=70,strict.width="cut")
```

```
## 'data.frame':    144 obs. of  17 variables:
##  $ ID                                 : num  1 2 3 4 4 4 5 6 7 8 ...
##  $ Parent.ID                          : num  NA NA NA 2 38 19 2 1 1 2..
##  $ Reinfection                        : logi  FALSE FALSE FALSE FALSE..
##  $ Infection_Date                     : chr  "2016/10/17" "2016/10/1"..
##  $ Infection_Time                     : chr  NA NA NA NA ...
##  $ Infection_Hours.since.start        : num  9.5 9.5 9.5 13.5 36.7 ...
##  $ Symptoms_Date                      : chr  NA NA NA NA ...
##  $ Symptoms_Time                      : chr  NA NA NA NA ...
##  $ Symptoms_Hours.since.start         : num  NA NA NA NA NA NA NA NA ..
##  $ End_Infection_Date                 : chr  "2016/10/17" "2016/10/1"..
##  $ End_Infection_Time                 : chr  NA NA NA NA ...
##  $ End_Infection_Hours.since.start    : num  14 15 16.7 18.6 NA ...
##  $ Onward_Infection_Hours.since.start: num  13.9 13.5 16.7 17.6 NA ...
##  $ Latent_Period_Hours                : num  4.43 4 7.18 4.08 NA ...
##  $ Incubation_Period_Hours            : num  NA NA NA NA NA NA NA NA ..
##  $ Infectious_Period_Hours            : num  0.07 1.5 0 1.04 NA ...
##  $ Generation_Time_Hours              : num  NA NA NA 4 0.34 ...
```

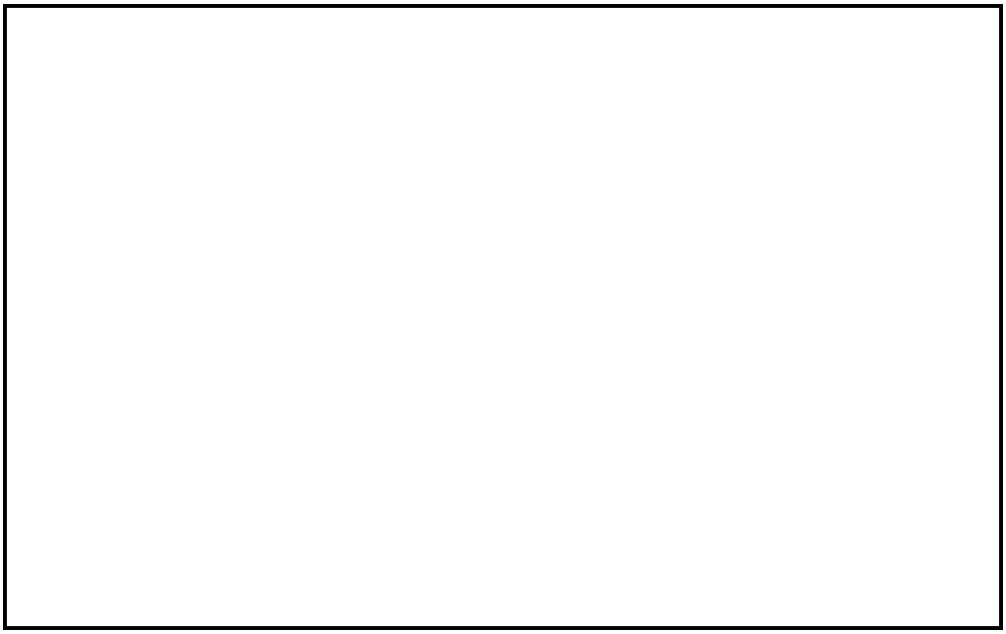
We can see that the loading of the data has resulted in a data frame object called *outbreak.dataset*, which includes the most important line list data from the excel spreadsheet. We can now use this data frame to further analyse the dataset.

Epidemic Time Series

Although we were able to plot the epidemic time series from excel, we carried this out by aggregating the time of infections and recoveries at daily intervals. This will potentially obscure some interesting infection dynamics. To view the full epidemic time series we can use the `epidemic_timeseries_plot` function:

```
outbreakteachR::epidemic_timeseries_plot(outbreak.dataset = outbreak.dataset)
```

- *Run the above command, and sketch it below.*
- *How does it look different to the excel plots?*
- *Did more infection occur at particular times in the day?*



Create an infection network

For the next task we will need to subset the collected data so that we are only looking at those cases that were not reinfections. Using this subset of the data we will then be able to plot a number of static images from the network that will enable us to see how the outbreak progressed in time.

```
# Convert the read outbreak dataset into only the non-reinfections
first_infection_list <- outbreakteachR::first_infection_list(outbreak.dataset)

# The above function has now sorted our data into a list of two dataframes.
# These data frames show the "linelist" and "contacts" data. We can now see
# that the linelist data only possesses information about contacts that were
# not reinfections:

unique(first_infection_list$linelist$Reinfection)
```

```
## [1] FALSE
```

Next we will create a plot of the outbreak that is time-orientated, i.e. the relative node positions represents the point in time that an individual was infected. To do this we can use the function `infection_network_plot` and providing as the first argument the *first_infection_list* we just created. To do this run the following code:

```
infection_network_plot(first_infection_list=first_infection_list,
                       time = TRUE, log = FALSE)
```

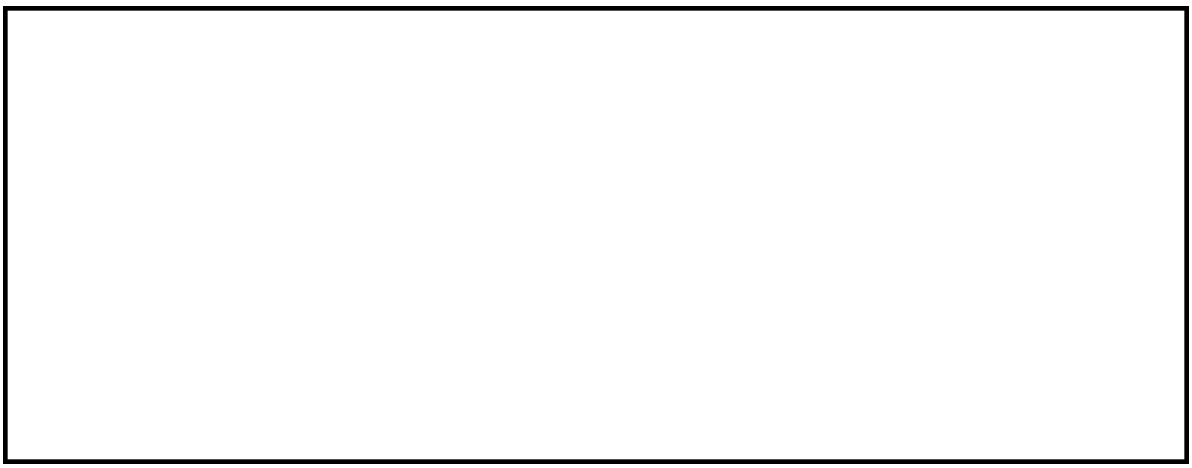
We can run this function again, however, this time set the last argument *log* to TRUE. This will modify the times that infection events occur by representing them on a log scale. To do this run the following code:

```
infection_network_plot(first_infection_list=first_infection_list,
                       time = TRUE, log = TRUE)
```

Lastly we can also plot the infection network, but not in a time-orientated way. This can be achieved by setting the argument *time* to FALSE. To do this run the following code:

```
infection_network_plot(first_infection_list=first_infection_list,
                       time = FALSE, log = TRUE)
```

- *Run the above commands one after another*
- *Which plot do you think was most informative for capturing the actual timing of the infection events?*
- *What sort of information about the outbreak was easiest to observe from the last plot?*
- *Do the initial seeds equally contribute to the final outbreak size?*
- *Do you think there is a more helpful way to visualise the infection network?*



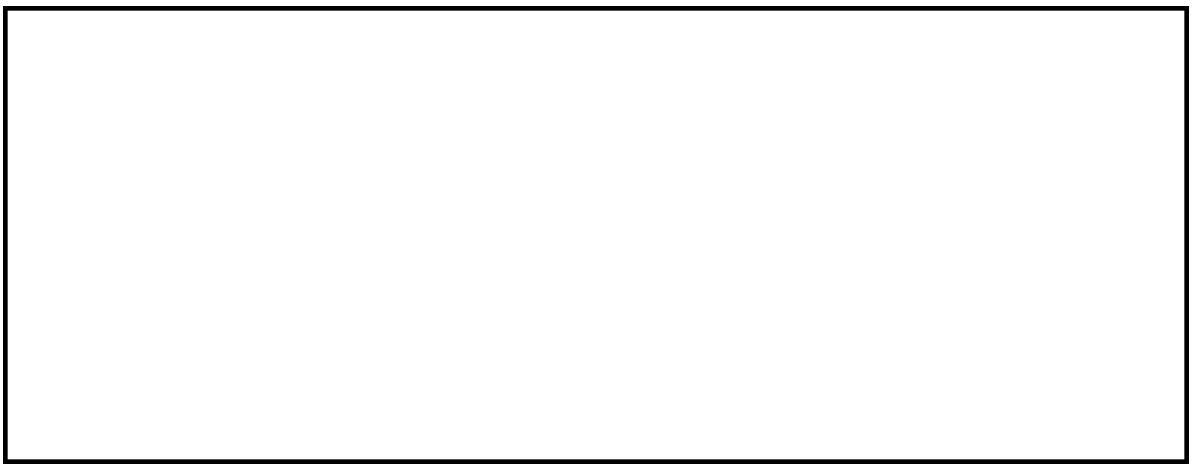
Animate the infection network

To improve some of the visualisation issues of the previous section it would be very helpful if we could animate the infection dynamics so that we could see how the outbreak spread throughout the population over time. To do this we can use the `animate_infection_network` function. To do this run the following code:

```
# Change the file parameter to wherever you want the html to be stored
animate_infection_network(first_infection_list=first_infection_list,
                          file=paste(getwd(),"/Network-Animation.html",sep=""),
                          year=2016)
```

When you run the above function the animated network will automatically load in a web browser and save the html page to the file path specified. The animated network can be viewed by clicking play, and the duration of the animation changed with the menu in the top left. To view the html for the 2016 outbreak output in a new tab please click here.

- *What time periods did most infection events occur?*
- *Was the timing of infection events equally distributed throughout the outbreak?*
- *Why would you expect the dynamics of the outbreak to not be uniform across each day?*



Simulated outbreaks

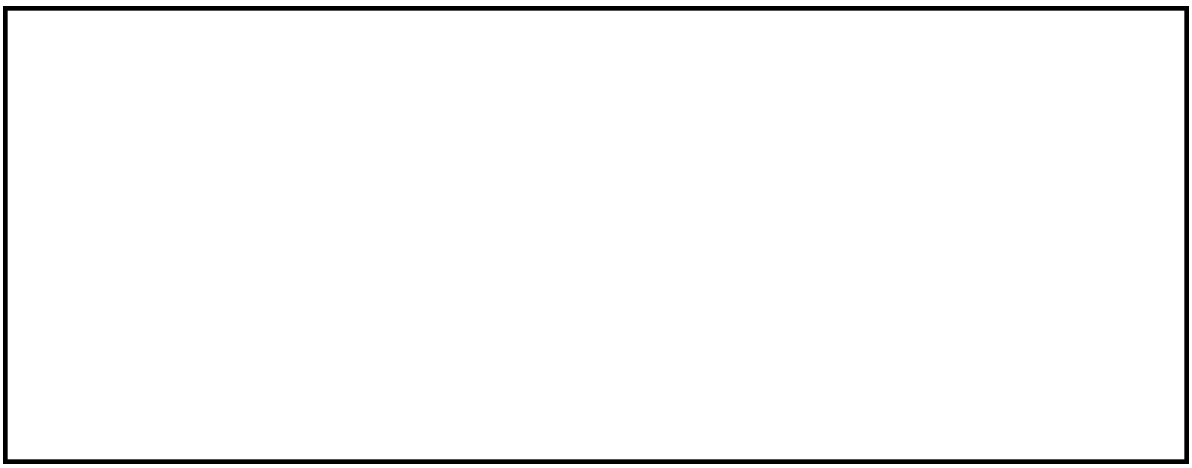
Mathematical models are a useful tool for understanding outbreak dynamics, and can be used to project the likely outcomes of an epidemic and help in the design of interventions. Within this last exercise we will use the following function to simulate 200 different outbreaks designed to replicate the observed outbreaks. These simulations are then summarised and plotted against the observed outbreak, showing the interquartile range of the simulations. The simulations are stochastic, which means that they incorporate random effects within the model.

(To give some more information into how the simulated outbreaks are modeled. For these simulations we assume each simulation is seeded with three infections at the same time as the observed outbreak, and has a total population size equal to the outbreak dataset. The number of secondary infections is then drawn from a Poisson distribution with a mean of 1.8, as was used in the practical. The time of each infection is also drawn from a Poisson with a mean equal to the observed mean generation times. The infected individual then recovers after a period of time, which is sampled from a Poisson with mean equal to the mean infectious period. If they do not infect anyone then they recover after a period of time drawn from a Poisson with mean equal to the mean recovery time for the observed outbreak. Simulations are then stopped after a period of time equal to the length of the observed outbreak.)

```
outbreakteachR::bootstrap_simulated_plot(
  first_infection_list = first_infection_list,
  outbreak.dataset = outbreak.dataset,
  lower.quantile=0.25,
  upper.quantile=0.75,
```

```
  include.observed = T,
  replicates = 200)
```

- *Did our simulations capture the shape of the observed outbreak?*
- *Why might our simulations fail to capture the observed outbreak's dynamics*
- *Why might we expect our simulation method to capture the outbreak's dynamics*
- *These are very open ended questions, so take some time to discuss these with eachother or the demonstrators*



Given the simulation methodology described above we can start to hypothesise about why the simulated outbreaks are delayed when compared to the observed. For example, we saw from the excel practical that the observed generation and recovery times from the outbreak do not follow a Poisson distribution. In fact they are more multimodal with a period of approximately 24 hours. As a result we might choose to conduct the simulations again but this time sampling the generation times, infectious periods and recovery times from those observed within the outbreak:

```
outbreakteachR::bootstrap_simulated_plot(
  first_infection_list = first_infection_list,
  outbreak.dataset = outbreak.dataset,
  lower.quantile = 0.25,
  upper.quantile = 0.75,
  include.observed = T,
  replicates = 200,
  sampling = TRUE)
```

This simulation should be much better, with the timing of the infection peak much closer to the observed data. This sampling method however still fails to perfectly capture the temporal dynamics exhibited. There are many reasons why this is, but we can begin to understand why this is by simply seeing how many infections there are that occurred on day 1, which is much higher than would be expected given the initial 3 seeds and the generation times that we sampled.

Conclusions

Hopefully the above tutorial has shown how the data visualisation can be a helpful tool to analyse the infection dynamics within an outbreak. We have also very briefly touched on how mathematical models can be used to simulate outbreaks, and how these can be used to analyse its infection dynamics. Lastly, we have covered briefly how to install R packages and use them to extend the range of tools available to us within R.