R4PDE

Table of contents

About

R for Plant Disease Epidemiology (R4PDE) is a book project in the development stage. It is based on the teaching notes of my graduate course - FIP 602 - Plant Disease Epidemiology, offered every year for students of the Graduate Program in Plant Pathology of the Universidade Federal de Viçosa.

This book is for those interested in studying and modelling plant disease epidemics using R programming. Here, I provide context and showcase several methods for describing, visualizing and analyzing epidemic data collected over time and/or space.

Users should have have a minimum knowledge of R programming. I make use of custom functions as well as several general and a few specific R packages for conducting the most common tasks related with the analysis of plant disease epidemiology data, in particular epifitter and epiphy. In most part, I use data and try to reproduce the several analyses shown in the book *The study of Plant Disease Epidemics* (Madden et al. 2017a)

1 Temporal analysis

1.1 Introduction

A key understanding of the epidemics relates to the knowledge of rates and patterns. Epidemics can be viewed as dynamic systems that change their state as time goes. The first and simplest way to characterize such changes in time is to produce a graphical plot called disease progress curve (DPC). This curve can be obtained as long as the intensity of the disease (y) in the host population is assessed sequentially in time (t).

A DPC summarizes the interaction of the three main components of the disease triangle occurring during the epidemic. The curves can vary greatly in shape according to variations in each of the components, in particular due to management practices that alter the course of the epidemics and for which the goal is to stop disease increase. We can create a dataframe in R for a single DPC and make a plot using ggplot. By convention we use `t` for time and `y` for disease intensity, expressed in proportion (0 to 1).

Firstly, let's load the essential R packages and set up the environment.

```r
library(tidyverse) # essential packages
library(cowplot) # for themes
theme_set(theme_minimal_grid()) # set global theme
```

There are several ways to create a dataframe in R. I like to use the `tribble` function as below. The entered data will be assigned to a dataframe called `dpc`.

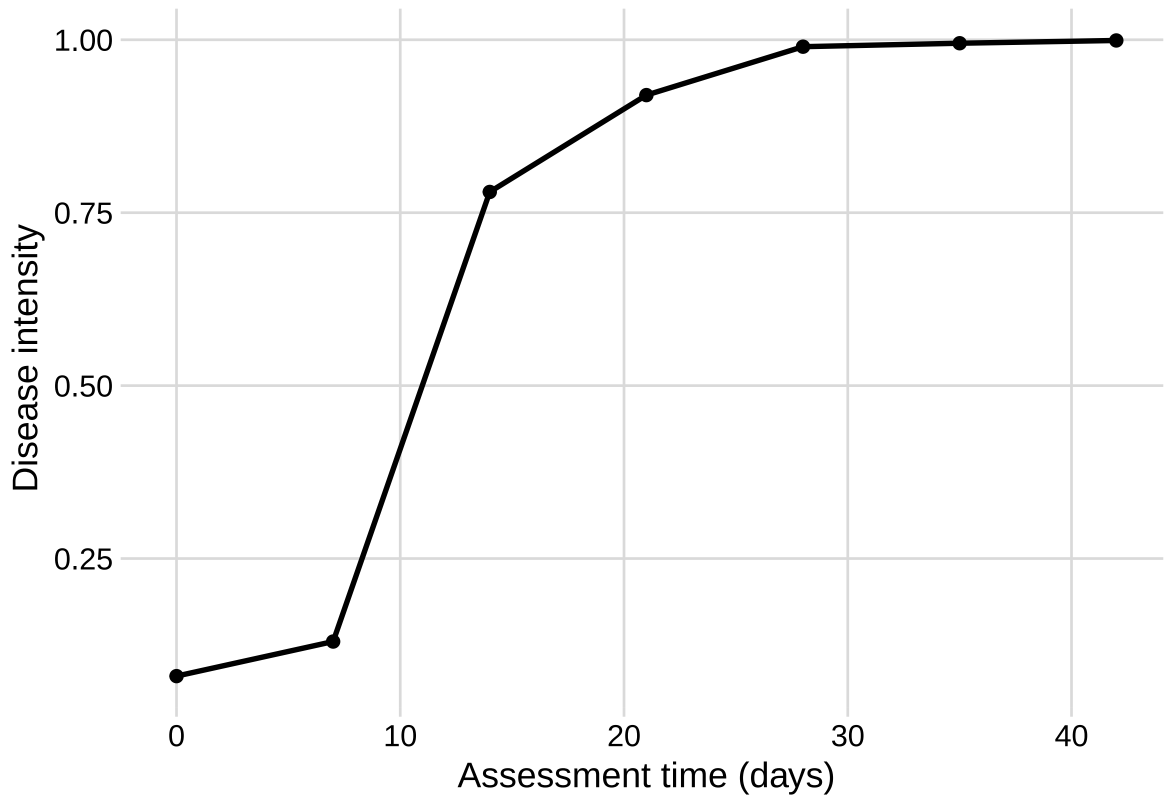
```
dpc <-
  tribble(
   ~t,   ~`y`,
   0,   0.08,
   7,   0.13,
  14,   0.78,
  21,   0.92,
  28,   0.99,
  35,  0.995,
  42,  0.999,
  )
```

Now the plot

Note

Note that I use the pipe operator `%>%` in my programming to express a sequence of multiple operations in a more intuitive way.

```
dpc %>%
  ggplot(aes(t, y)) +
  geom_point(size = 2)+
  geom_line(size = 1)+
  labs(x = "Assessment time (days)",
       y = "Disease intensity")
```



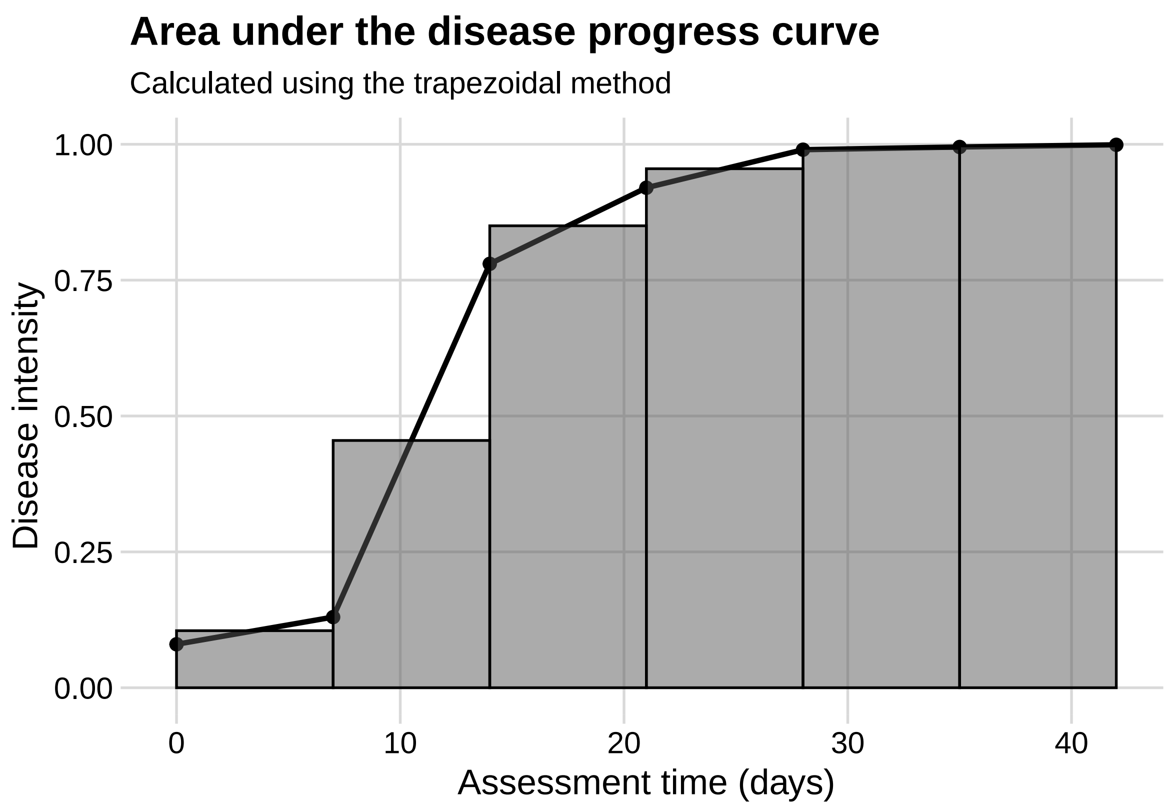
1.2 Curve descriptors: AUDPC

The depiction and analysis of disease progress curves can provide useful information for gaining understanding of the underlying epidemic process. The curves are extensively used to evaluate how disease control measures affect epidemics. When characterizing DPCs, a researcher may be interested in describing and comparing epidemics that result from different treatments, or simply in their variations as affected by changes in environment, host or pathogen.

The precision and complexity of the analysis of progress curve data depends on the objective of the study. In general, the goal is to synthesize similarities and different among epidemics based on common descriptors of the disease progress curves. For example, the simple appraisal of the disease intensity at any time during the course of the epidemic should be sufficient for certain situations. Furthermore, a few descriptors can be extracted including the epidemic duration, the initial and maximum disease, and the area under the disease progress curve (AUDPC).

The AUDPC summarizes the "total measure of disease stress" and is largely used to compare epidemics (Jeger and Viljanen-Rollinson 2001). The most common approach to calculate AUDPC is the trapezoidal method, which splits the disease progress curves into a series of rectangles, calculating the area of each of them and then summing the areas. Let's extend the plot code to show those rectangles using the `annotate` function.

```r
dpc %>%
  ggplot(aes(t, y)) +
  geom_point(size = 2)+
  geom_line(size = 1)+
  labs(x = "Assessment time (days)",
       y = "Disease intensity",
       title ="Area under the disease progress curve",
       subtitle = "Calculated using the trapezoidal method")+
  annotate("rect", xmin = dpc$t[1], xmax = dpc$t[2],
           ymin = 0, ymax = (dpc$y[1]+ dpc$y[2])/2,
           color = "black", alpha = 0.5)+
   annotate("rect", xmin = dpc$t[2], xmax = dpc$t[3],
            ymin = 0, ymax = (dpc$y[2]+ dpc$y[3])/2,
            color = "black", alpha = 0.5)+
   annotate("rect", xmin = dpc$t[3], xmax = dpc$t[4],
            ymin = 0, ymax = (dpc$y[3]+ dpc$y[4])/2,
            color = "black", alpha = 0.5)+
   annotate("rect", xmin = dpc$t[4], xmax = dpc$t[5],
            ymin = 0, ymax = (dpc$y[4]+ dpc$y[5])/2,
            color = "black", alpha = 0.5)+
   annotate("rect", xmin = dpc$t[5], xmax = dpc$t[6],
            ymin = 0, ymax = (dpc$y[5]+ dpc$y[6])/2,
            color = "black", alpha = 0.5)+
   annotate("rect", xmin = dpc$t[6], xmax = dpc$t[7],
            ymin = 0, ymax = (dpc$y[6]+ dpc$y[7])/2,
            color = "black", alpha = 0.5)
```



In R, we can obtain the AUDPC for the DPC we created earlier using the `AUDPC` function offered by the *epifitter* package. Because we are using the percent data, we need to set the argument `y_proportion = FALSE`. The function returns the absolute AUDPC. If one is interested in relative AUDPC, the argument `type` should be set to `"relative"`. There is also the alternative to AUDPC, the area under the disease progress stairs (AUDPS) (Simko and Piepho 2012).

```r
library(epifitter)
AUDPC(dpc$t, dpc$y,
      y_proportion = FALSE)
```

```
[1] 30.4815
```

```r
# The relative AUDPC
AUDPC(dpc$t, dpc$y,
      y_proportion = FALSE,
      type = "relative")
```

```
[1] 0.0072575
```

```
# To calculate AUDPS, the alternative to AUDPC
AUDPS(dpc$t, dpc$y,
      y_proportion = FALSE)
```

```
[1] 34.258
```

Mathematical models can be fitted to the DPC data to express epidemic progress in terms of rates and absolute/relative quantities. The latter can be accomplished using population dynamics (or growth-curve) models for which the estimated parameters are usually meaningful biologically and appropriately describe epidemics that do not decrease in disease intensity. By fitting an appropriate model to the progress curve data, another set of parameters is available to the researcher when attempting to represent, understand or compare epidemics.

1.3 Population dynamics models

The family of models that describe the growth of epidemics, hence population dynamics model, are known as deterministic models of continuous time (Madden et al. 2017b). These models are usually fitted to DPC data to obtain two or more biologically meaningful parameters. Here, these models and their formulations are shown using R scripts to simulate the theoretical curves for each model.

1.3.1 Non-flexible models

These population dynamics models require at least two parameters, hence they are known as non-flexible, as opposed to the flexible ones for which there are at least one additional (third) parameter.

Following the convention proposed by (Madden et al. 2017b) in their book "The study of plant disease epidemics":

- time is represented by t
- disease intensity by y
- the rate of change in y between two time units is represented by $\frac{dy}{dt}$

Now we can proceed and learn which non-flexible models exist and for which situation they are more appropriate.

1.3.1.1 Exponential

The differential equation for the exponential model is given by

$\frac{dy}{dt} = r_E.y$,

where r_E is the apparent infection rate (subscript E for this model) (sensu Vanderplank) and y is the disease intensity. Biologically, this formulation suggests that diseased plants, or y, and r_E at each time contribute to disease increase. The value of $\frac{dy}{dt}$ is minimal when $y = 0$ and increases exponentially with the increase in y.

The integral for the exponential model is given by

$y = y_0 e^{r_E t}$,

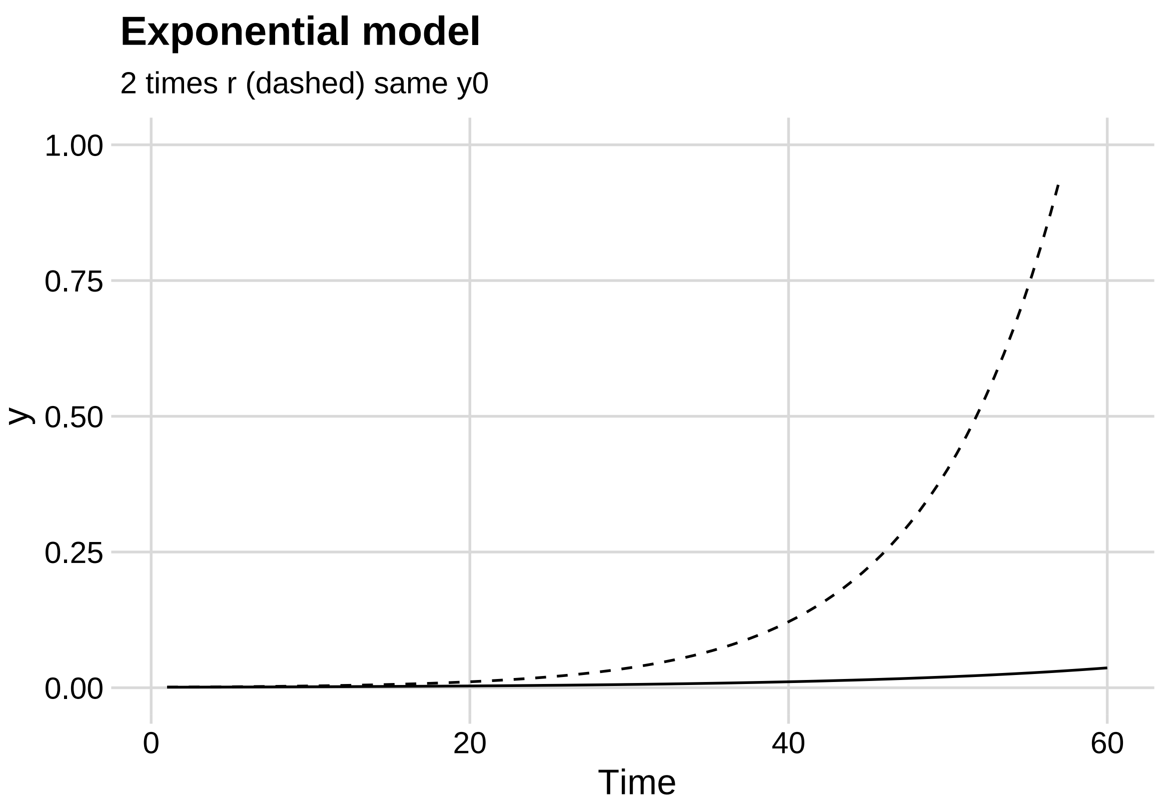
where $y0$ is and r are obtained via estimation. Let's simulate two curves by varying r while fixing $y0$ and varying the latter while fixing r_E. We produce the two plots in *ggplot* and add the predicted curve using the 'stat_function'. But first, we need to define values for the two model parameters. Further modifications to these values will be handled directly in the simulation (e.g. doubling infection rate, reducing initial inoculum by half, etc.).

```r
y0 <- 0.001
r <- 0.06
tmax <- 60 # maximum duration t of the epidemics
dat <- data.frame(t = seq(1:tmax), y = seq(0:1)) # define the axes
```

In the plot below, note that the infection rate in one curve was doubled ($r = 0.12$)

```r
dat %>%
  ggplot(aes(t, y)) +
  stat_function(fun = function(t) y0 * exp(r * t), linetype = 1) +
  stat_function(fun = function(t) y0 * exp(r * 2 * t), linetype = 2) +
  ylim(0, 1) +
  labs(
    title = "Exponential model",
    subtitle = "2 times r (dashed) same y0",
    x = "Time"
  )
```

```
Warning: Removed 5 row(s) containing missing values (geom_path).
```

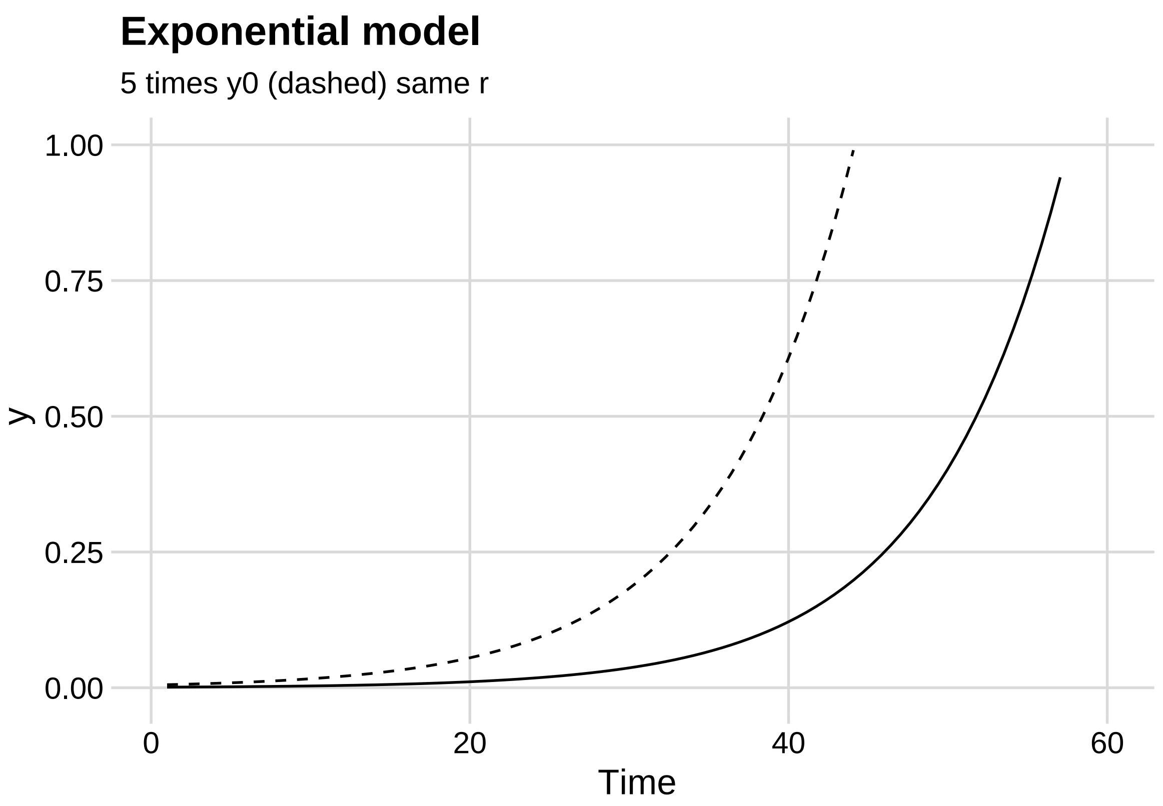


Now the inoculum was increased five times while using the same doubled rate.

```
dat %>%
  ggplot(aes(t, y)) +
  stat_function(fun = function(t) y0 * exp(r * 2 * t), linetype = 1) +
  stat_function(fun = function(t) y0 * 5 * exp(r * 2 * t), linetype = 2) +
  ylim(0, 1) +
  labs(title = "Exponential model", x = "Time",
       subtitle = "5 times y0 (dashed) same r")
```

```
Warning: Removed 5 row(s) containing missing values (geom_path).
```

```
Warning: Removed 27 row(s) containing missing values (geom_path).
```



1.3.1.2 Monomolecular

The differential of the monomolecular model is given by

$\frac{dy}{dt} = r_M(1-y)$

where now the r_M is the rate parameter of the monomolecular model and $(1-y)$ is the proportion of non-infected (healthy) individuals or host tissue. Note that $\frac{dy}{dt}$ is maximum when $y = 0$ and decreases when y approaches 1. Its decline is due to decrease in the proportion of individuals or healthy sites with the increase in y. Any inoculum capable of infecting the host will more likely land on infected individuals or sites.

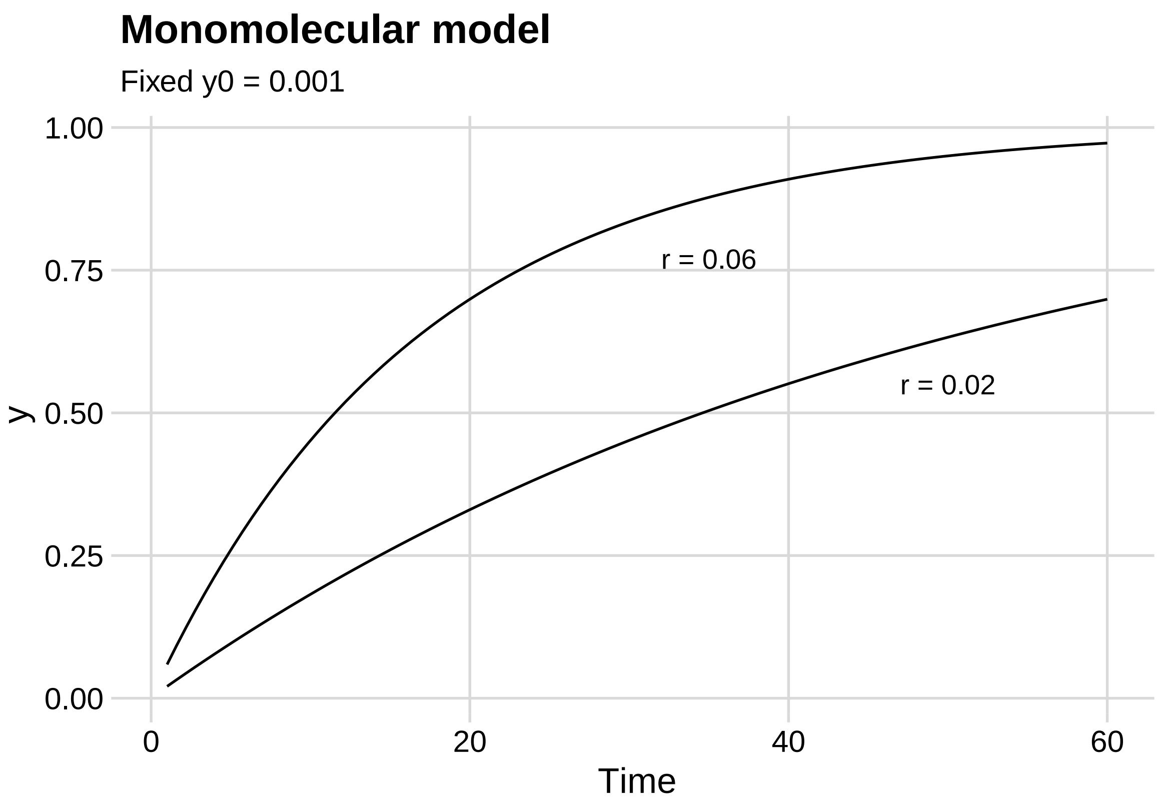
The integral of the monomolecular model is given by

$\frac{dy}{dt} = 1 - (1-y)e^{-r_M t}$

This model commonly describes the temporal patterns of the monocyclic epidemics. In those, the inoculum produced during the course of the epidemics do not contribute new infections. Therefore, different from the exponential model, disease intensity y does not affect the epidemics and so the absolute rate is proportional to $(1-y)$.

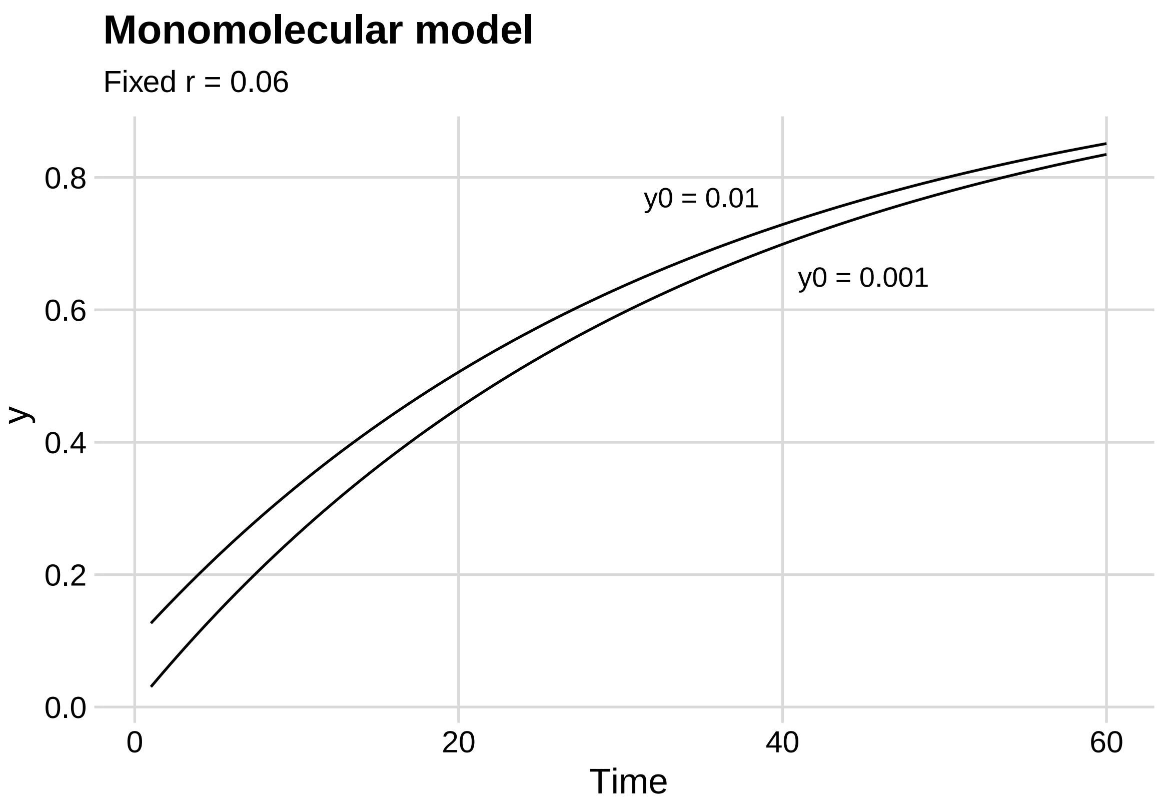
Let's simulate two monomolecular curve with different rate parameters where one is one third of the other.

```r
dat %>%
  ggplot(aes(t, y)) +
  stat_function(fun = function(t) 1 - ((1 - y0) * exp(-r * t))) +
  stat_function(fun = function(t) 1 - ((1 - y0) * exp(-(r / 3) * t))) +
  labs(title = "Monomolecular model",
         subtitle = "Fixed y0 = 0.001", x = "Time"
      ) +
  annotate(geom = "text", x = 35, y = 0.77, label = "r = 0.06") +
  annotate(geom = "text", x = 50, y = 0.55, label = "r = 0.02")
```



Now inoculum was increased 100 times with the reduced rate.

```r
dat %>%
  ggplot(aes(t, y)) +
  stat_function(fun = function(t) 1 - ((1 - y0) * exp(-r / 2 * t))) +
  stat_function(fun = function(t) 1 - ((1 - (y0 * 100)) * exp(-r / 2 * t))) +
  labs(title = "Monomolecular model",
       subtitle = "Fixed r = 0.06", x = "Time") +
  annotate(geom = "text", x = 35, y = 0.77, label = "y0 = 0.01") +
  annotate(geom = "text", x = 45, y = 0.65, label = "y0 = 0.001")
```



1.3.1.3 Logistic

The logistic model is a more elaborated version of the two previous models as it incorporates the features of them both. Its differential is given by

$\frac{dy}{dt} = r_L.y.(1-y)$,

where r_L is the infection rate of the logistic model, y is the proportion of diseased individuals or host tissue and $(1-y)$ is the proportion of non-affected individuals or host area.

Biologically, y in its differential equation implies that $\frac{dy}{dt}$ increases with the increase in y (as in the exponential) because more disease means more inoculum. However, $(1 - y)$ leads to a decrease in $\frac{dy}{dt}$ when y approaches the maximum $y = 1$, because the proportion of healthy individuals or host area decreases (as in the monomolecular). Therefore, $\frac{dy}{dt}$ is minimal at the onset of the epidemics, reaches a maximum when $y/2$ and declines until $y = 1$.

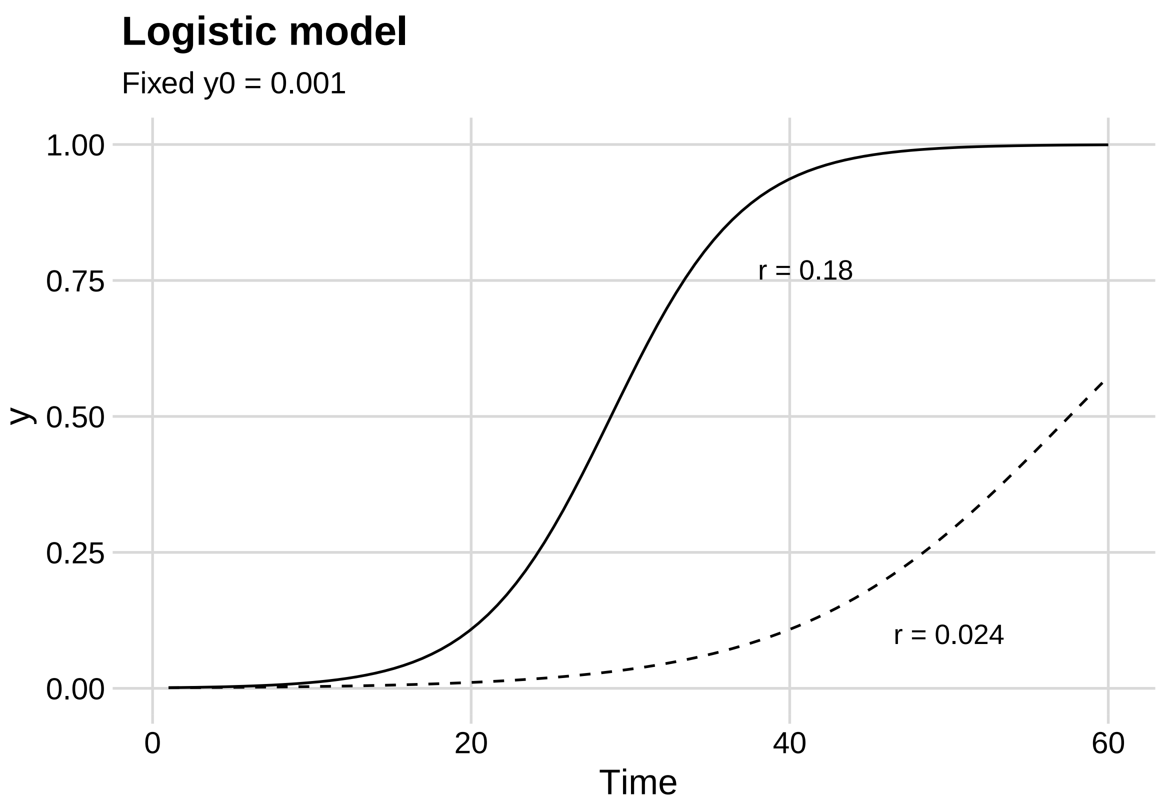
The integral is given by

$y = \frac{1}{1+(1-y_0).e^{-r.t}}$,

where r_L is the apparent infection rate of the logistic model and $y0$ is the disease intensity at $t = 0$. This model provides a good fit to polycyclic epidemics.

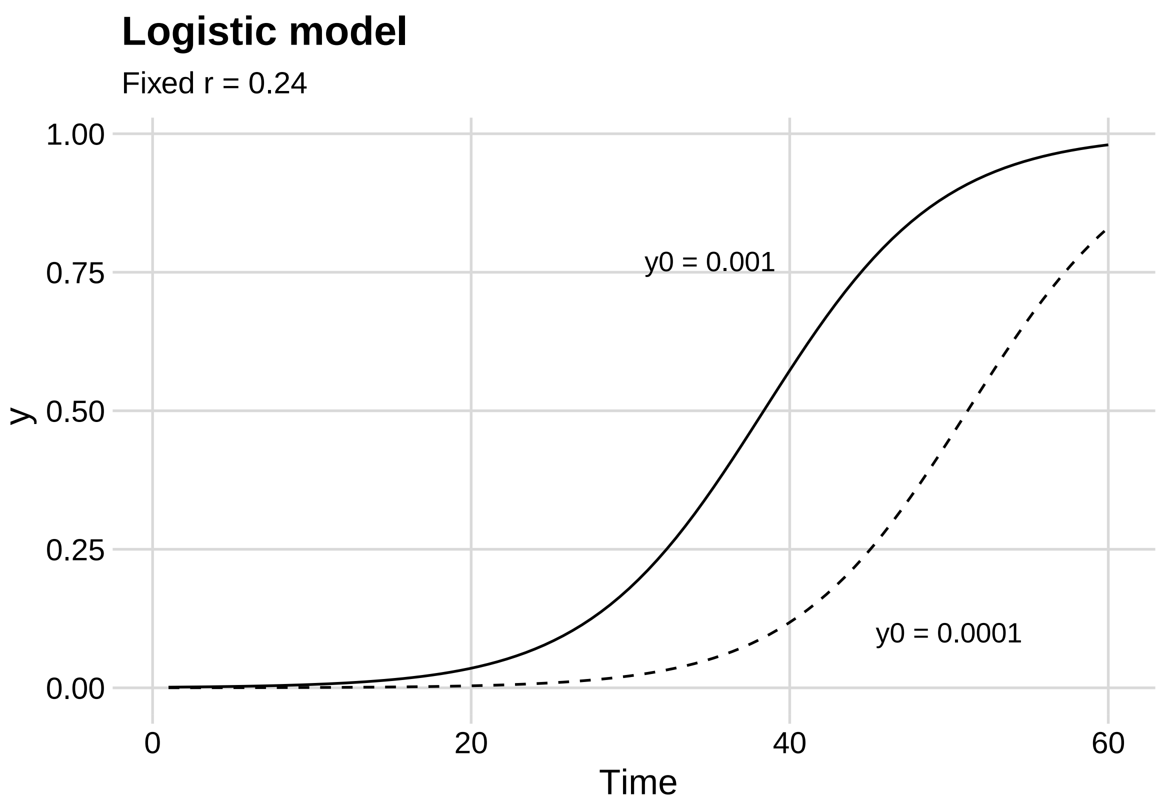
Let's check two curves where in one the infection rate is double while keeping the same initial inoculum.

```
dat %>%
  ggplot(aes(t, y)) +
  stat_function(
    linetype = 2,
    fun = function(t) 1 / (1 + ((1 - y0) / y0) * exp(-r * 2 * t))
  ) +
  stat_function(fun = function(t) 1 / (1 + ((1 - y0) / y0) * exp(-r * 4 * t))) +
  labs(title = "Logistic model", subtitle = "Fixed y0 = 0.001", x = "Time") +
  annotate(geom = "text", x = 41, y = 0.77, label = "r = 0.18") +
  annotate(geom = "text", x = 50, y = 0.10, label = "r = 0.024")
```



Now the inoculum is reduced 10 times for a same infection rate.

```
dat %>%
  ggplot(aes(t, y)) +
  stat_function(
    linetype = 2,
    fun = function(t) 1 / (1 + ((1 - (y0 / 10)) / (y0 / 10)) * exp(-r * 3 * t))
  ) +
  stat_function(fun = function(t) 1 / (1 + ((1 - y0) / y0) * exp(-r * 3 * t))) +
  labs(title = "Logistic model", subtitle = "Fixed r = 0.24", x = "Time") +
  annotate(geom = "text", x = 35, y = 0.77, label = "y0 = 0.001") +
  annotate(geom = "text", x = 50, y = 0.10, label = "y0 = 0.0001")
```



1.3.1.4 Gompertz

The Gompertz model is similar to the logistic and also provides a very good fit to several polycyclic diseases. The differential equation is given by

$\frac{dy}{dt} = r_G.[ln(1) - ln(y)]$

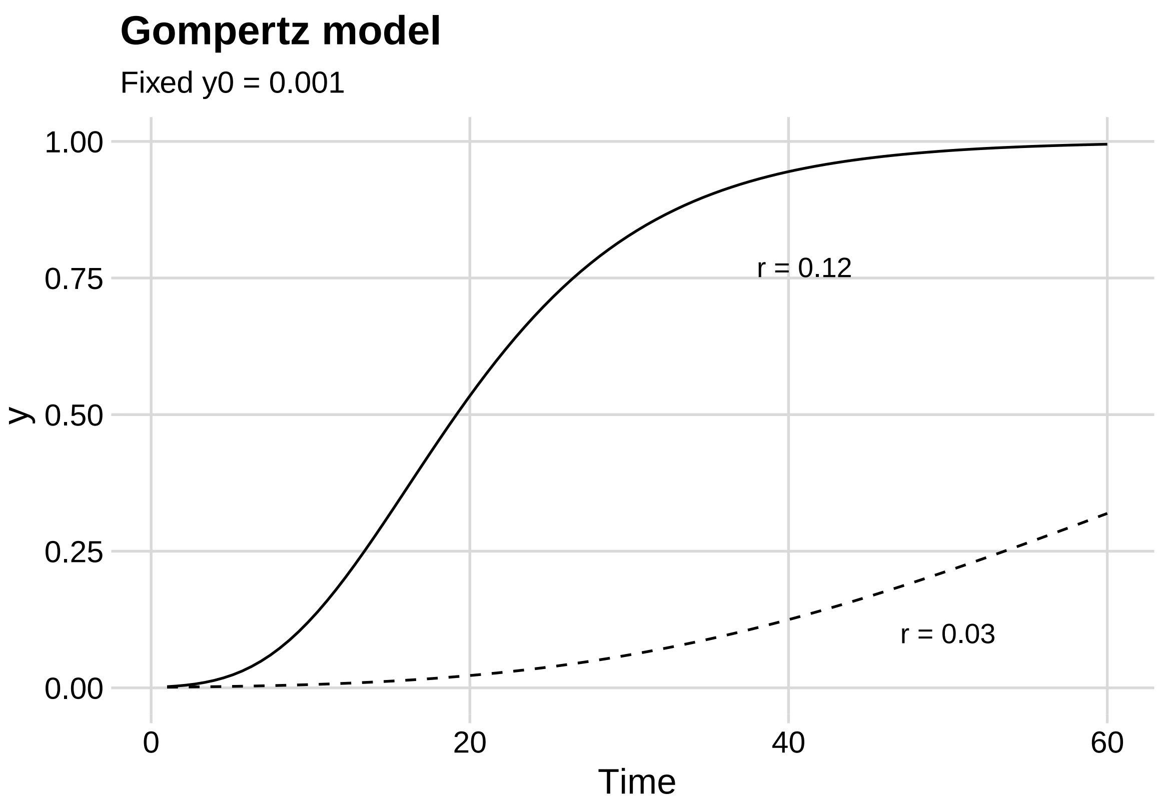
Differently from the logistic, the variable representing the non-infected individuals or host area is $-ln(y)$. The integral equation is given by

$y = e^{(ln(y0)).e^{-r_G.t)}}$,

where r_G is the apparent infection rate for the Gompertz models and y_0 is the disease intensity at $t = 0$.

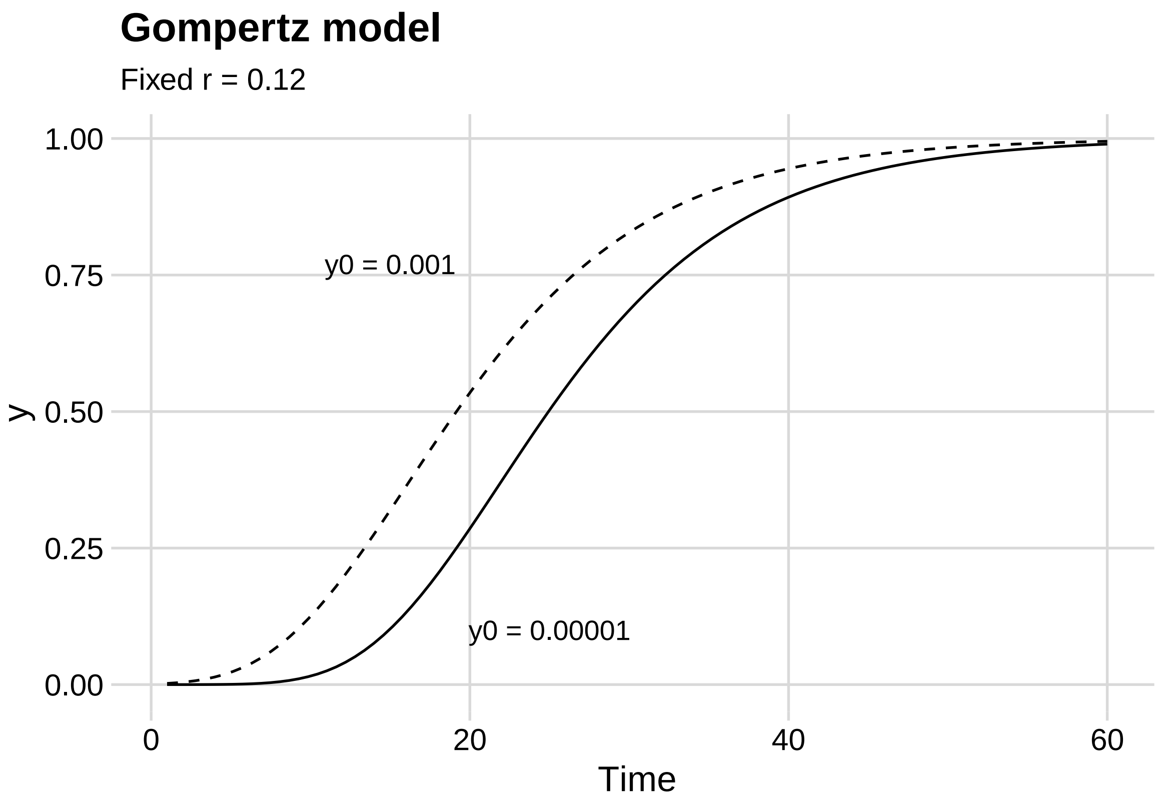
Let's check curves for two rates.

```
dat %>%
  ggplot(aes(t, y)) +
  stat_function(
    linetype = 2,
    fun = function(t) exp(log(y0) * exp(-r/2 * t))
  ) +
  stat_function(fun = function(t) exp(log(y0) * exp(-r*2 * t))) +
  labs(title = "Gompertz model", subtitle = "Fixed y0 = 0.001", x = "Time") +
  annotate(geom = "text", x = 41, y = 0.77, label = "r = 0.12") +
  annotate(geom = "text", x = 50, y = 0.10, label = "r = 0.03")
```



And those when inoculum was reduced thousand times.

```r
dat %>%
  ggplot(aes(t, y)) +
  stat_function(
    linetype = 2,
    fun = function(t) exp(log(y0) * exp(-r*2 * t))
  ) +
  stat_function(fun = function(t) exp(log(y0/1000) * exp(-r*2 * t))) +
  labs(title = "Gompertz model", subtitle = "Fixed r = 0.12", x = "Time") +
  annotate(geom = "text", x = 15, y = 0.77, label = "y0 = 0.001") +
  annotate(geom = "text", x = 25, y = 0.10, label = "y0 = 0.00001")
```



1.4 Fitting models to data

Next, you will learn how to fit models to multiple actual disease progress curves (DPCs) data obtained from the literature. I will demonstrate how to fit and select the models using the *epifitter* package. A few user friendly functions will help us decide which model to choose to obtain the parameters of interest and further compare the epidemics.

To illustrate, I will use two datasets available from Chapter 3 from the book, *Study of Plant Disease Epidemics* (Madden et al. 2017b). In the book, SAS codes are presented to perform a few analysis. We then provide an alternative code for performing similar analysis, although not perfectly reproducing the results from the book.

1.4.1 Non-replicated epidemics

We will compare three DPCs of the incidence of tobacco etch, a virus disease, in peppers. Evaluations of incidence were evaluated at a 7-day interval, up to 49 days.The data are available in chapter 4 (page 93). Let's input the data manually and create a data frame. First column is the assessment time and the other columns correspond to the treatments, called groups in the book, from 1 to 3.

1.4.2 Entering data

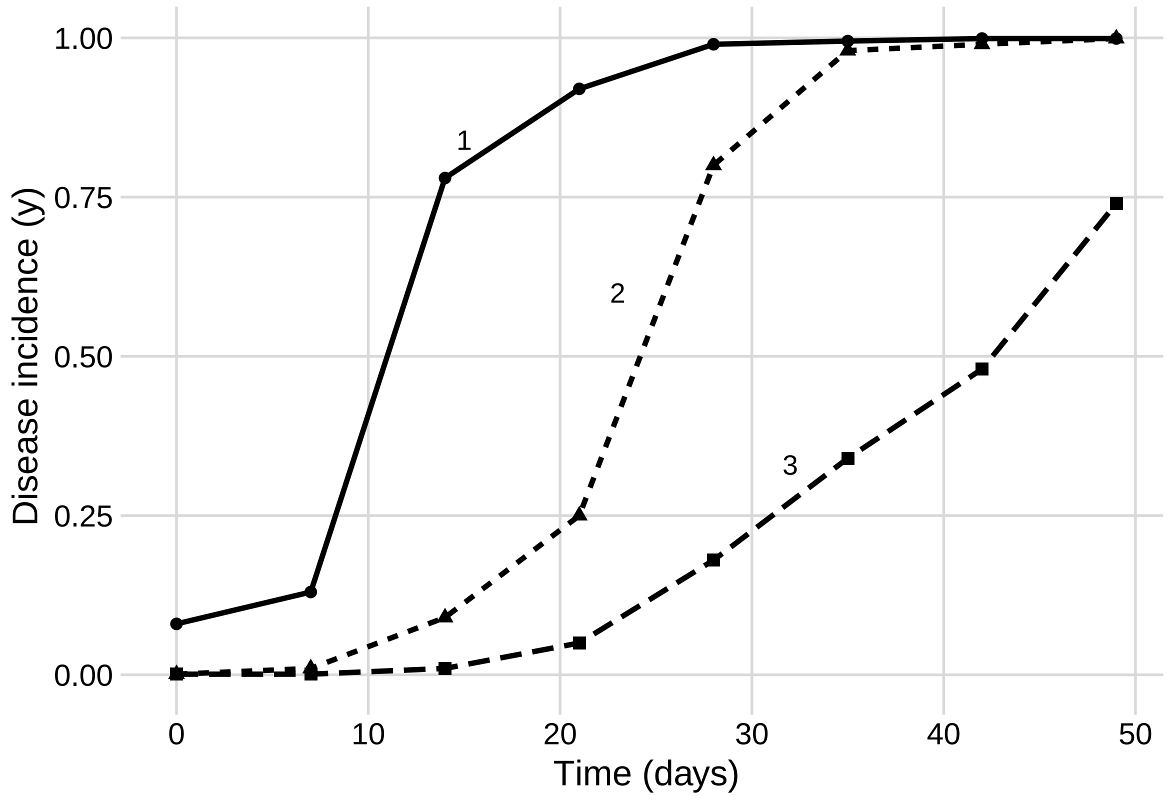
```r
pepper <-
  tribble(
   ~t,  ~`1`,  ~`2`,  ~`3`,
   0,  0.08, 0.001, 0.001,
   7,  0.13,  0.01, 0.001,
  14,  0.78,  0.09,  0.01,
  21,  0.92,  0.25,  0.05,
  28,  0.99,   0.8,  0.18,
  35, 0.995,  0.98,  0.34,
  42, 0.999,  0.99,  0.48,
  49, 0.999, 0.999,  0.74
  )
```

1.4.3 Visualize the DPCs

Before proceeding with model selection and fitting, let's visualize the three epidemics. The code below reproduces quite exactly the top plot of Fig. 4.15 ((Madden et al. 2017b) page 94). The appraisal of the curves might give us a hint on which models are the best candidates.

Because the data was entered in the wide format (each DPCs in a different columns) we need to reshape it to the long format. The `pivot_longer` function will do the job of reshaping from wide to long format so we can finally use the `ggplot` function to produce the plot.

```
pepper %>%
  pivot_longer(2:4, names_to ="treat", values_to = "inc") %>%
  ggplot (aes(t, inc,
              linetype = treat,
              shape = treat,
              group = treat))+
  geom_point(size =2)+
  geom_line(size = 1)+
  annotate(geom = "text", x = 15, y = 0.84, label = "1")+
  annotate(geom = "text", x = 23, y = 0.6, label = "2")+
  annotate(geom = "text", x = 32, y = 0.33, label = "3")+
  labs(y = "Disease incidence (y)",
       x = "Time (days)")+
  theme(legend.position = "none")
```



Most of the three curves show a sigmoid shape with the exception of group 3 that resembles an exponential growth, not reaching the maximum value, and thus suggesting an incomplete epidemic. We can easily eliminate the monomolecular and exponential models and decide on the other two non-flexible models: logistic or Gompertz. To do that, let's proceed to model fitting and evaluate the statistics for supporting a final decision. There are two modeling approaches for model fitting in epifitter: the **linear** or **nonlinear** parameter-estimation methods.

1.4.4 Fitting: single epidemics

Among the several options offered by *epifitter* we start with the simplest one, which is fit a model to a single epidemics using the linear regression approach. For such, the `fit_lin()` requires two arguments: time (`time`) and disease intensity (`y`) each one as a vector stored or not in a dataframe.

Since we have three epidemics, `fit_lin()` will be use three times. The function produces a list object with six elements. Let's first look at the `Stats` dataframe of each of the three lists named `epi1` to `epi3`.

```
library(epifitter)
epi1 <- fit_lin(time = pepper$t,
                y = pepper$`1` )
epi1$Stats
```

```
                CCC r_squared    RSE
Gompertz     0.9848    0.9700 0.5911
Monomolecular 0.9838    0.9681 0.5432
Logistic     0.9782    0.9572 0.8236
Exponential  0.7839    0.6447 0.6705
```

```
epi2 <- fit_lin(time = pepper$t,
  y = pepper$`2` )
epi2$Stats
```

```
                CCC r_squared    RSE
Logistic     0.9962    0.9924 0.4524
Gompertz     0.9707    0.9431 0.8408
Monomolecular 0.9248    0.8601 1.0684
Exponential  0.8971    0.8134 1.2016
```

```
epi3 <- fit_lin(time = pepper$t,
  y = pepper$`3` )
epi3$Stats
```

```
                    CCC r_squared    RSE
Logistic      0.9829    0.9665 0.6045
Gompertz      0.9825    0.9656 0.2263
Exponential   0.9636    0.9297 0.7706
Monomolecular 0.8592    0.7531 0.2534
```

The statistics of the model fit confirms our initial guess that the predictions by the logistic or the Gompertz are closer to the observations than predictions by the other models. There is no much difference between them based on these statistics. However, to pick one of the models, it is important to inspect the curves with the observed and predicted values to check which model is best for all curves.

1.4.5 Fitting: multiple epidemics

Before looking at the prediction, let's use another handy function that allows us to simultaneously fit the models to multiple DPC data. Different from `fit_lin()`, `fit_multi()` requires the data to be structured in the long format where there is a column specifying each of the epidemics.

Let's then create a new data set called `pepper2` using the data transposing functions of the *tidyr* package.

```r
pepper2 <- pepper %>%
  pivot_longer(2:4, names_to ="treat", values_to = "inc")
```

Now we fit the models to all DPCs. Note that the name of the variable indicating the DPC code needs to be informed in `strata_cols` argument.

```r
epi_all <- fit_multi(
  time_col = "t",
  intensity_col = "inc",
  data = pepper2,
  strata_cols = "treat",
  nlin = FALSE
)
```

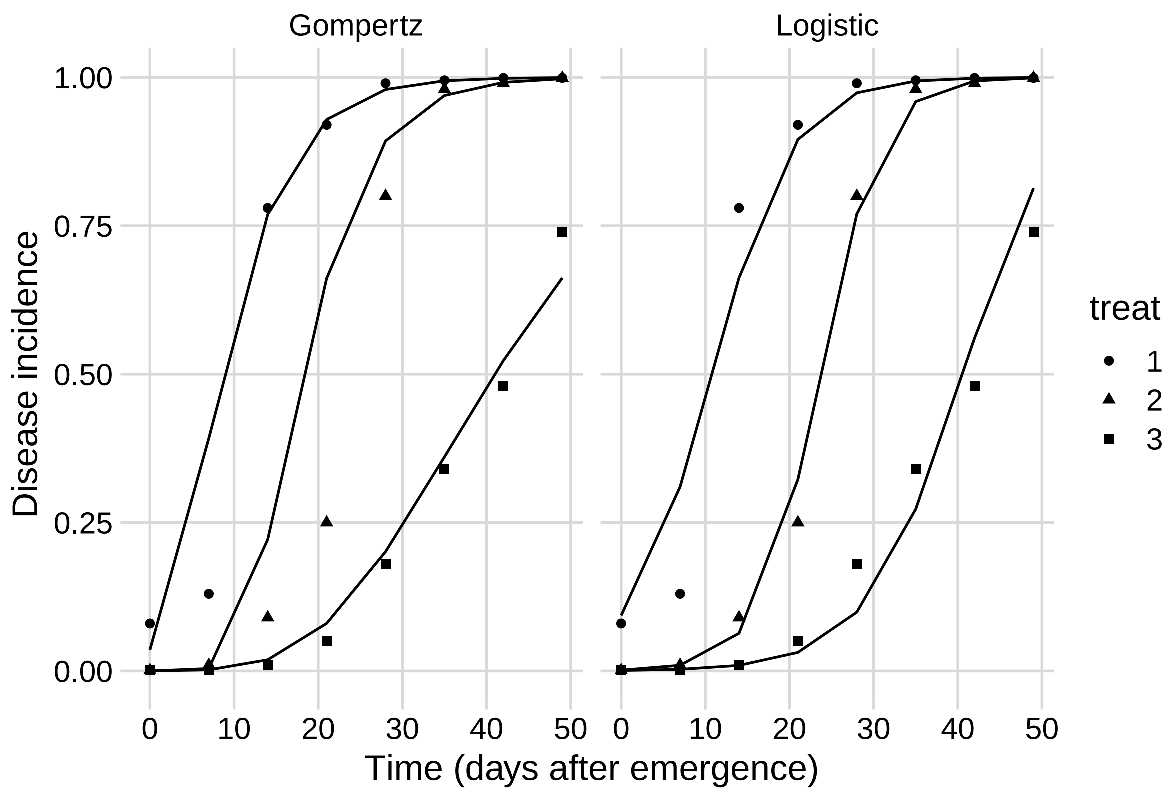
Now let's select the statistics of model fitting. Again, *Epifitter* ranks the models based on the CCC (the higher the better) but it is important to check the RSE as well - the lower the better. In fact, the RSE is more important when the goal is prediction.

```
epi_all$Parameters %>%
  select(treat, model, best_model, RSE, CCC)
```

```
   treat         model best_model       RSE       CCC
1      1      Gompertz          1 0.5911056 0.9847857
2      1 Monomolecular          2 0.5431977 0.9838044
3      1      Logistic          3 0.8235798 0.9781534
4      1   Exponential          4 0.6705085 0.7839381
5      2      Logistic          1 0.4523616 0.9961683
6      2      Gompertz          2 0.8407922 0.9707204
7      2 Monomolecular          3 1.0683633 0.9247793
8      2   Exponential          4 1.2015809 0.8971003
9      3      Logistic          1 0.6045243 0.9829434
10     3      Gompertz          2 0.2262550 0.9824935
11     3   Exponential          3 0.7705736 0.9635747
12     3 Monomolecular          4 0.2533763 0.8591837
```

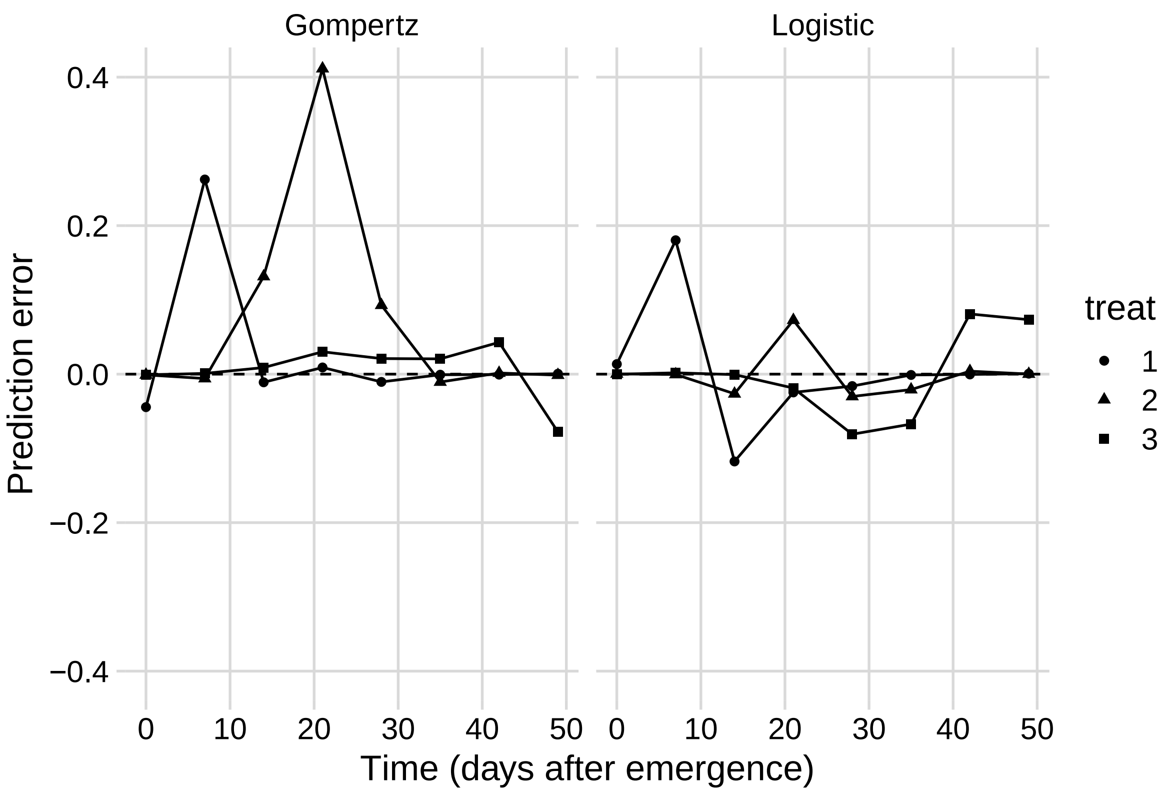
To be more certain about our decision, let's advance to the final step which is to produce the plots with the observed and predicted values for each assessment time by calling the `Data` dataframe of the ‘`epi_all` list.

```
epi_all$Data %>%
 filter(model %in% c("Gompertz", "Logistic")) %>%
  ggplot(aes(time, predicted, shape = treat)) +
  geom_point(aes(time, y)) +
  geom_line() +
  facet_wrap(~ model) +
 coord_cartesian(ylim = c(0, 1)) + # set the max to 0.6
  labs(
    y = "Disease incidence",
    x = "Time (days after emergence)"
  )
```



Overall, the logistic model seems a better fit for all the curves. Let's produce a plot with the prediction error versus time.

```
epi_all$Data %>%
 filter(model %in% c("Gompertz", "Logistic")) %>%
  ggplot(aes(time, predicted -y, shape = treat)) +
  geom_point() +
  geom_line() +
  geom_hline(yintercept = 0, linetype =2)+
  facet_wrap(~ model) +
 coord_cartesian(ylim = c(-0.4, 0.4)) + # set the max to 0.6
  labs(
    y = "Prediction error",
    x = "Time (days after emergence)"
  )
```



The plots above confirms the logistic model as good fit overall because the errors for all epidemics combined are more scattered around the non-error line.

```
    epi_all$Parameters %>%
      filter(model == "Logistic") %>%
      select(treat, y0, y0_ci_lwr, y0_ci_upr, r, r_ci_lwr, r_ci_upr
  )
```

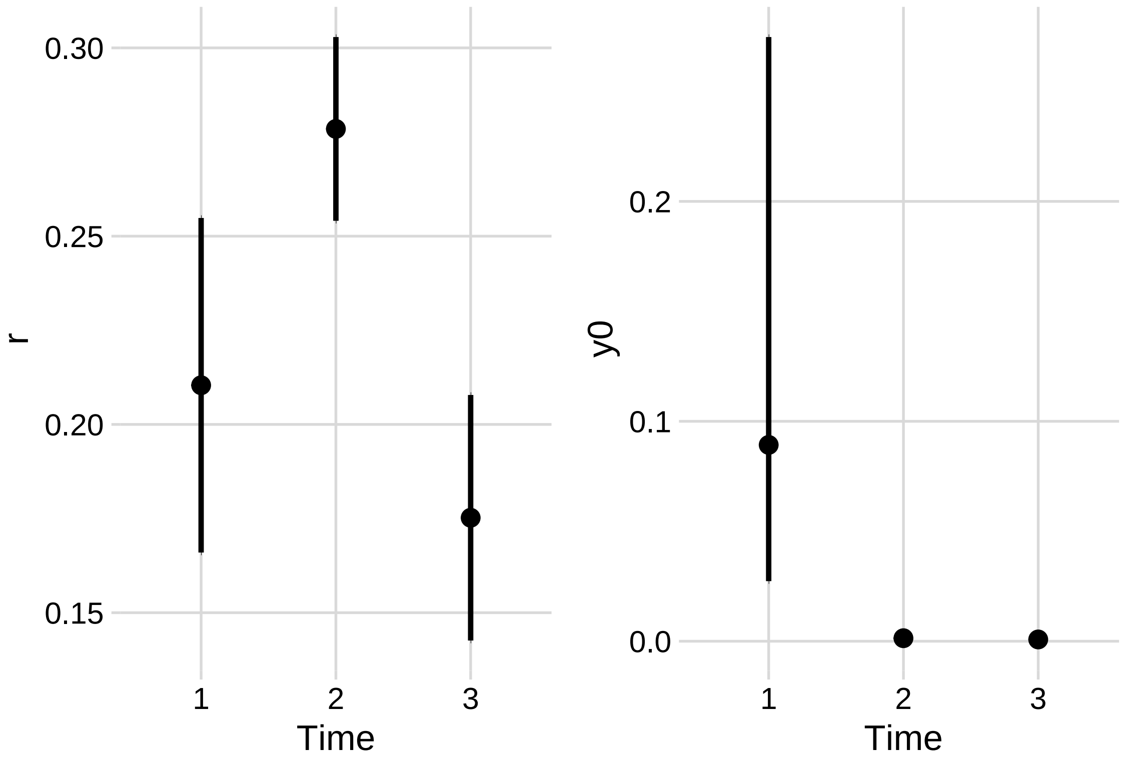
```
  treat           y0    y0_ci_lwr   y0_ci_upr         r  r_ci_lwr  r_ci_upr
1     1 0.0935037690 0.0273207272 0.274728744 0.2104047 0.1659824 0.2548270
2     2 0.0013727579 0.0006723537 0.002800742 0.2784814 0.2540818 0.3028809
3     3 0.0008132926 0.0003131745 0.002110379 0.1752146 0.1426077 0.2078215
```

We can produce a plot for visual inference on the differences in the parameters.

```r
p1 <- epi_all$Parameters %>%
  filter(model == "Logistic") %>%
  ggplot(aes(treat, r)) +
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = r_ci_lwr, ymax = r_ci_upr),
    width = 0,
    size = 1
  ) +
  labs(
    x = "Time",
    y = "r"
  )

p2 <- epi_all$Parameters %>%
  filter(model == "Logistic") %>%
  ggplot(aes(treat, 1 - exp(-y0))) +
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = y0_ci_lwr, ymax = y0_ci_upr),
    width = 0,
    size = 1
  ) +
  labs(
    x = "Time",
    y = "y0"
  )

library(patchwork)
```

```

Attaching package: 'patchwork'

The following object is masked from 'package:cowplot':

    align_plots
```

```r
p1 | p2
```



1.4.6 Designed experiments

In this next section, we will work with disease data collected over time in the same plot unit (also called repeated measures) from a designed experiment for evaluating and comparing treatment effects.

Again, we will use a dataset of progress curves shown in page 98 (Madden et al. 2017b). The curves represent the incidence of soybean plants symptomatic for bud blight caused by tobacco streak virus. Four treatments (different planting dates) were evaluated in randomized complete block design with four replicates. There are four assessment in time for each curve. The data was stored as a csv file and will be loaded using `read_csv()` function and stored as dataframe called `budblight`.

1.4.6.1 Loading data

```
budblight <- read_csv("data/bud-blight-soybean.csv")
```

```
-- Column specification --------------------------------------------------------
cols(
  treat = col_character(),
  time = col_double(),
  block = col_double(),
  y = col_double()
)
```

Let's have a look at the first six rows of the dataset and check the data type for each column. There is an additional column representing the replicates, called block.

```
head(budblight)
```

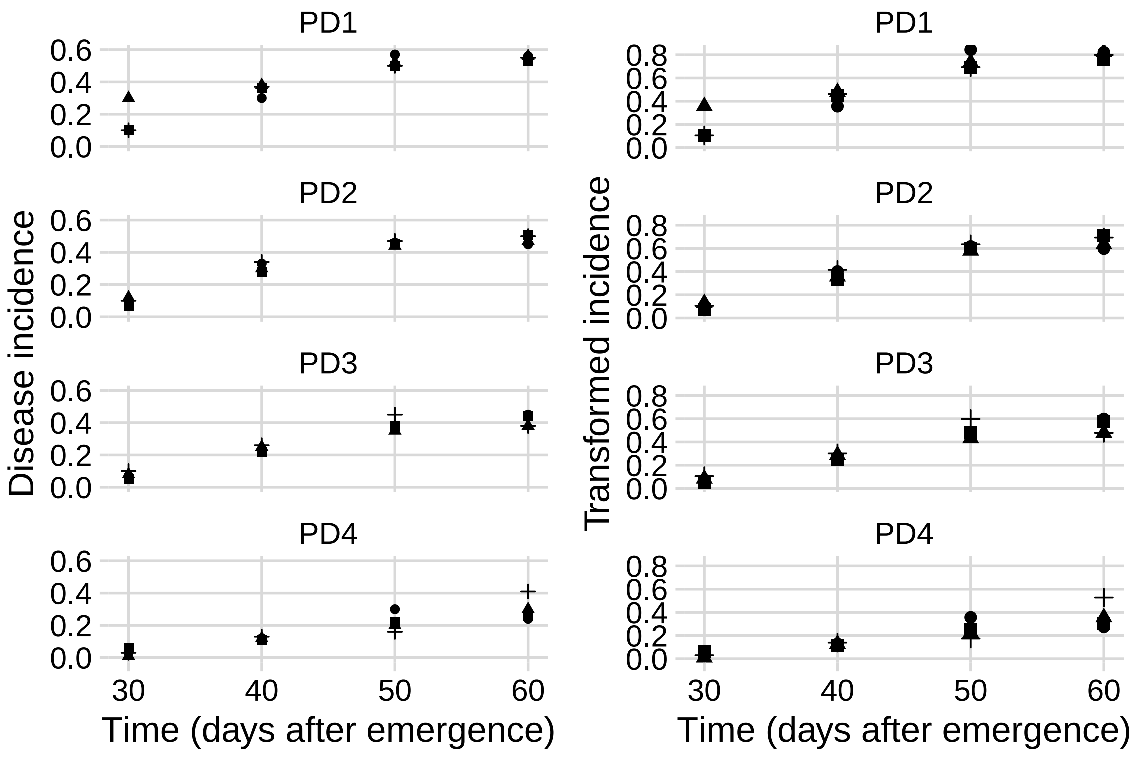
```
# A tibble: 6 x 4
  treat  time block     y
  <chr> <dbl> <dbl> <dbl>
1 PD1      30     1  0.1
2 PD1      30     2  0.3
3 PD1      30     3  0.1
4 PD1      30     4  0.1
5 PD1      40     1  0.3
6 PD1      40     2  0.38
```

####Visualizing the DPCs

Let's have a look at the curves and produce a combo plot figure similar to Fig. 4.17 of the book, but without the line of the predicted values.

```
p3 <- budblight %>%
  ggplot(aes(
    time, y,
    group = block,
    shape = factor(block)
  )) +
  geom_point(size = 1.5) +
  ylim(0, 0.6) +
  theme(legend.position = "none")+
  facet_wrap(~treat, ncol =1)+
  labs(y = "Disease incidence",
       x = "Time (days after emergence)")
```

```r
p4 <- budblight %>%
  ggplot(aes(
    time, log(1 / (1 - y)),
    group = block,
    shape = factor(block)
  )) +
  geom_point(size = 2) +
  facet_wrap(~treat, ncol = 1) +
  theme(legend.position = "none")+
  labs(y = "Transformed incidence", x = "Time (days after emergence)")

p3 | p4
```



1.4.6.2 Model fitting

Remember that the first step in model selection is the visual appraisal of the curve data linearized with the model transformation. In the case the curves represent complete epidemics (close to 100%) appraisal of the absolute rate (difference in y between two times) over time is also helpful.

For the treatments above, it looks like the curves are typical of a monocyclic disease (the case of soybean bud blight), for which the monomolecular is usually a good fit, but other models are also possible as well. For this exercise, we will use both the linear and the nonlinear estimation method.

1.4.6.2.1 Linear regression

For convenience, we use the `fit_multi()` to handle multiple epidemics. The function returns a list object where a series of statistics are provided to aid in model selection and parameter estimation. We need to provide the names of columns (arguments): assessment time (`time_col`), disease incidence (`intensity_col`), and treatment (`strata_cols`).

```
lin1 <- fit_multi(
  time_col = "time",
  intensity_col = "y",
  data = budblight,
  strata_cols = "treat",
  nlin = FALSE
)
```

Let's look at how well the four models fitted the data. Epifitter suggests the best fitted model (1 to 4, where 1 is best) for each treatment. Let's have a look at the statistics of model fitting.

```
  lin1$Parameters %>%
    select(treat, best_model, model, CCC, RSE)
```

```
   treat best_model         model       CCC        RSE
1    PD1          1 Monomolecular 0.9348429 0.09805661
2    PD1          2      Gompertz 0.9040182 0.22226189
3    PD1          3      Logistic 0.8711178 0.44751963
4    PD1          4   Exponential 0.8278055 0.36124036
5    PD2          1 Monomolecular 0.9547434 0.07003116
6    PD2          2      Gompertz 0.9307192 0.17938711
7    PD2          3      Logistic 0.9062012 0.38773023
8    PD2          4   Exponential 0.8796705 0.32676216
9    PD3          1 Monomolecular 0.9393356 0.06832499
10   PD3          2      Gompertz 0.9288436 0.17156394
11   PD3          3      Logistic 0.9085414 0.39051075
```

```
12    PD3           4   Exponential 0.8896173 0.33884790
13    PD4           1      Gompertz 0.9234736 0.17474422
14    PD4           2 Monomolecular 0.8945962 0.06486949
15    PD4           3      Logistic 0.8911344 0.52412586
16    PD4           4   Exponential 0.8739618 0.49769642
```

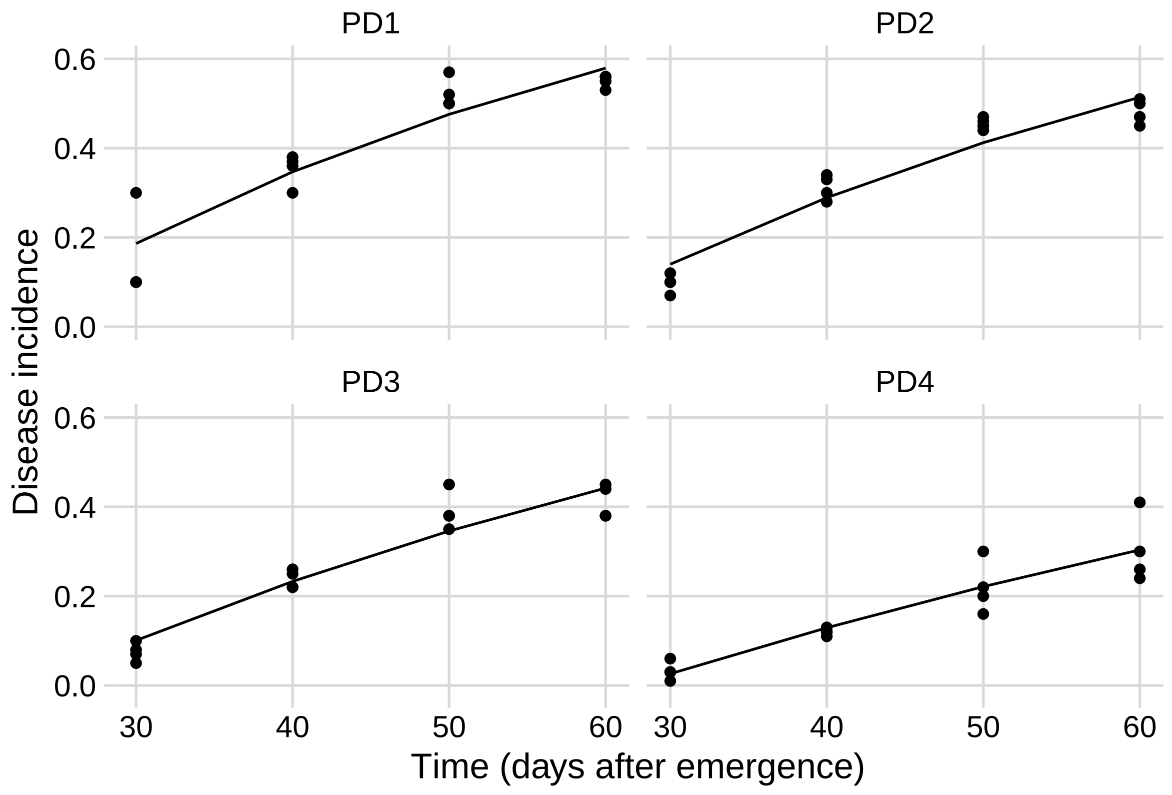
And now we extract values for each parameter estimated from the fit of the monomolecular model.

```
    lin1$Parameters %>%
    filter(model == "Monomolecular") %>%
    select(treat, y0, r)
```

```
  treat         y0          r
1   PD1 -0.5727700 0.02197351
2   PD2 -0.5220593 0.01902952
3   PD3 -0.4491365 0.01590586
4   PD4 -0.3619898 0.01118047
```

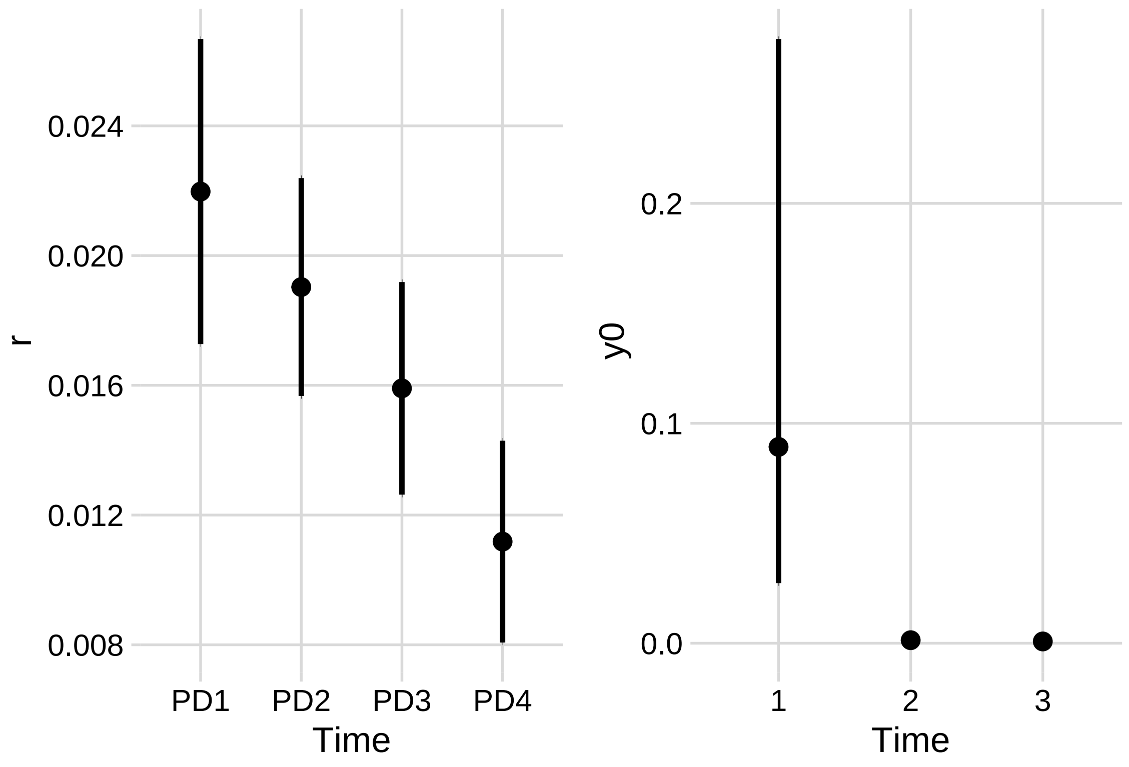
Now we visualize the fit of the monomolecular model (using `filter` function - see below) to the data together with the observed data and then reproduce the right plots in Fig. 4.17 from the book.

```
  lin1$Data %>%
    filter(model == "Monomolecular") %>%
    ggplot(aes(time, predicted)) +
    geom_point(aes(time, y)) +
    geom_line(size = 0.5) +
    facet_wrap(~treat) +
    coord_cartesian(ylim = c(0, 0.6)) + # set the max to 0.6
    labs(
      y = "Disease incidence",
      x = "Time (days after emergence)"
    )
```



Now we can plot the means and respective 95% confidence interval of the apparent infection rate (r) and initial inoculum (y_0) for visual inference.

```r
p5 <- lin1$Parameters %>%
  filter(model == "Monomolecular") %>%
  ggplot(aes(treat, r)) +
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = r_ci_lwr, ymax = r_ci_upr),
    width = 0,
    size = 1
  ) +
  labs(
    x = "Time",
    y = "r"
  )

p6 <- lin1$Parameters %>%
  filter(model == "Monomolecular") %>%
  ggplot(aes(treat, 1 - exp(-y0))) +
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = y0_ci_lwr, ymax = y0_ci_upr),
    width = 0,
    size = 1
  ) +
  labs(
    x = "Time",
    y = "y0"
  )

p5 | p2
```



1.4.6.2.2 Non-linear regression

To estimate the parameters using the non-linear approach, we repeat the same arguments in the `fit_multi` function, but include an additional argument `nlin` set to `TRUE`.

```r
nlin1 <- fit_multi(
  time_col = "time",
  intensity_col = "y",
  data = budblight,
  strata_cols = "treat",
  nlin = TRUE
)
```

```
Warning in log(y0/1): NaNs produced

Warning in log(y0/1): NaNs produced

Warning in log(y0/1): NaNs produced
```

Let's check statistics of model fit.

```
nlin1$Parameters %>%
  select(treat, model, CCC, RSE, best_model)
```

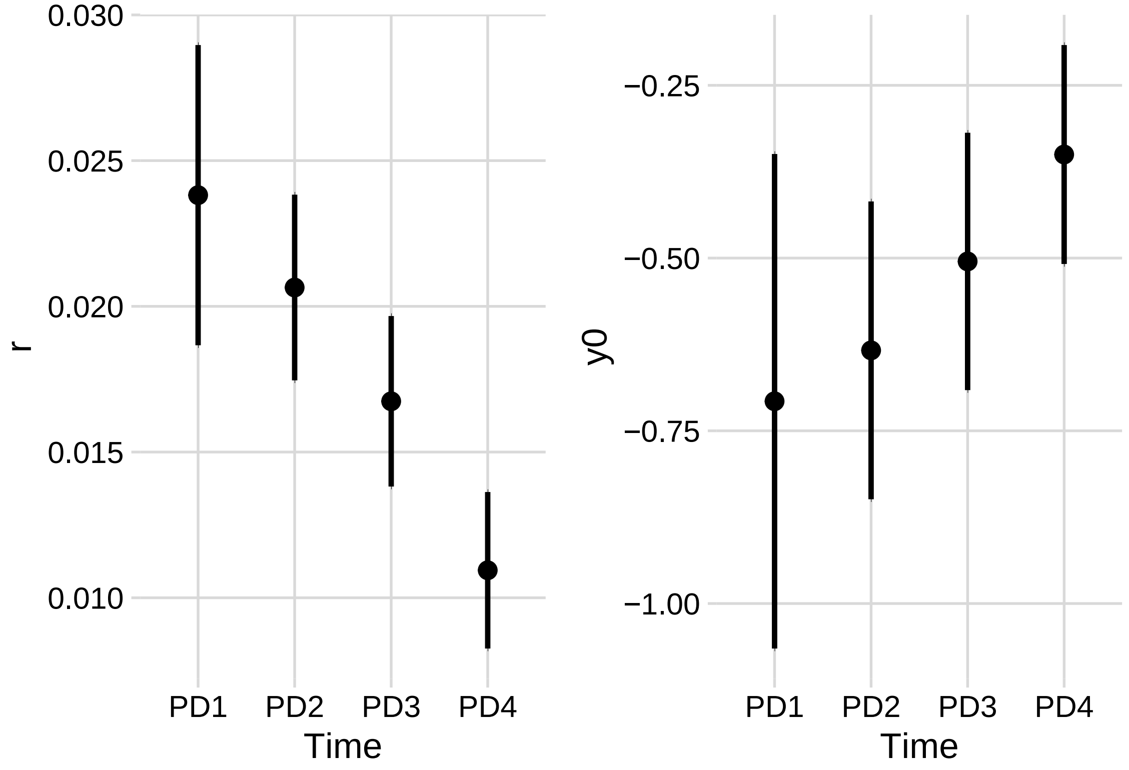
```
   treat         model       CCC        RSE best_model
1    PD1 Monomolecular 0.9382991 0.06133704          1
2    PD1      Gompertz 0.9172407 0.06986307          2
3    PD1      Logistic 0.8957351 0.07700720          3
4    PD1   Exponential 0.8544194 0.08799512          4
5    PD2 Monomolecular 0.9667886 0.04209339          1
6    PD2      Gompertz 0.9348370 0.05726761          2
7    PD2      Logistic 0.9077857 0.06657793          3
8    PD2   Exponential 0.8702365 0.07667322          4
9    PD3 Monomolecular 0.9570853 0.04269129          1
10   PD3      Gompertz 0.9261609 0.05443852          2
11   PD3      Logistic 0.8997106 0.06203037          3
12   PD3   Exponential 0.8703443 0.06891021          4
13   PD4 Monomolecular 0.9178226 0.04595409          1
14   PD4      Gompertz 0.9085579 0.04791331          2
15   PD4      Logistic 0.8940731 0.05083336          3
16   PD4   Exponential 0.8842437 0.05267415          4
```

And now we obtain the two parameters of interest. Note that the values are not the sames as those estimated using linear regression, but they are similar and highly correlated.

```
  nlin1$Parameters %>%
    filter(model == "Monomolecular") %>%
    select(treat, y0, r)
```

```
  treat         y0          r
1   PD1 -0.7072562 0.02381573
2   PD2 -0.6335713 0.02064629
3   PD3 -0.5048763 0.01674209
4   PD4 -0.3501234 0.01094368
```

```r
p7 <- nlin1$Parameters %>%
  filter(model == "Monomolecular") %>%
  ggplot(aes(treat, r)) +
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = r_ci_lwr, ymax = r_ci_upr),
    width = 0,
    size = 1
  ) +
  labs(
    x = "Time",
    y = "r"
  )

p8 <- nlin1$Parameters %>%
  filter(model == "Monomolecular") %>%
  ggplot(aes(treat, y0)) +
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = y0_ci_lwr, ymax = y0_ci_upr),
    width = 0,
    size = 1
  ) +
  labs(
    x = "Time",
    y = "y0"
  )

p7 | p8
```


0.030
0.025
0.020
0.015
0.010
r
PD1
PD2
PD3
PD4
Time
−0.25
−0.50
−0.75
−1.00
y0
PD1
PD2
PD3
PD4
Time

2 Spatial gradients

2.1 Models

When modeling disease gradients, the distance is represented by x, a continuous variable which can be expressed by various units (cm, m, km, etc). The gradient models, similar to the population dynamics models (disease progress) are of the **deterministic** type. The difference is that, for disease progress curves, disease intensity tends to increase with increasing time, while in disease gradients the disease intensity tends to decrease with increasing distance from the source of inoculum. Two models are most commonly fitted to data on disease gradients. More details about these models can be obtained it this tutorial.

2.1.1 Exponential model

The exponential model is also known as Kiyosawa & Shiyomi model. The differential of the exponential model is given by

$\frac{dy}{dx} = -b_E.y$,

where b_E is the exponential form of the rate of decline and y is the disease intensity. This model suggests that y (any disease intensity) is greater close to the source of inoculum, or at the distance zero. The integral form of the model is given by

$y = a.e^{-b.x}$,

where a is the disease intensity at the distance zero and b is the rate of decline, in this case negative because disease intensity decreases with the increase of the distance from inoculum source. Let's make a plot for two disease gradients of varying parameters for this model.

First we need to load essential packages for programming, customizing the outputs and defining a global ggplot theme.

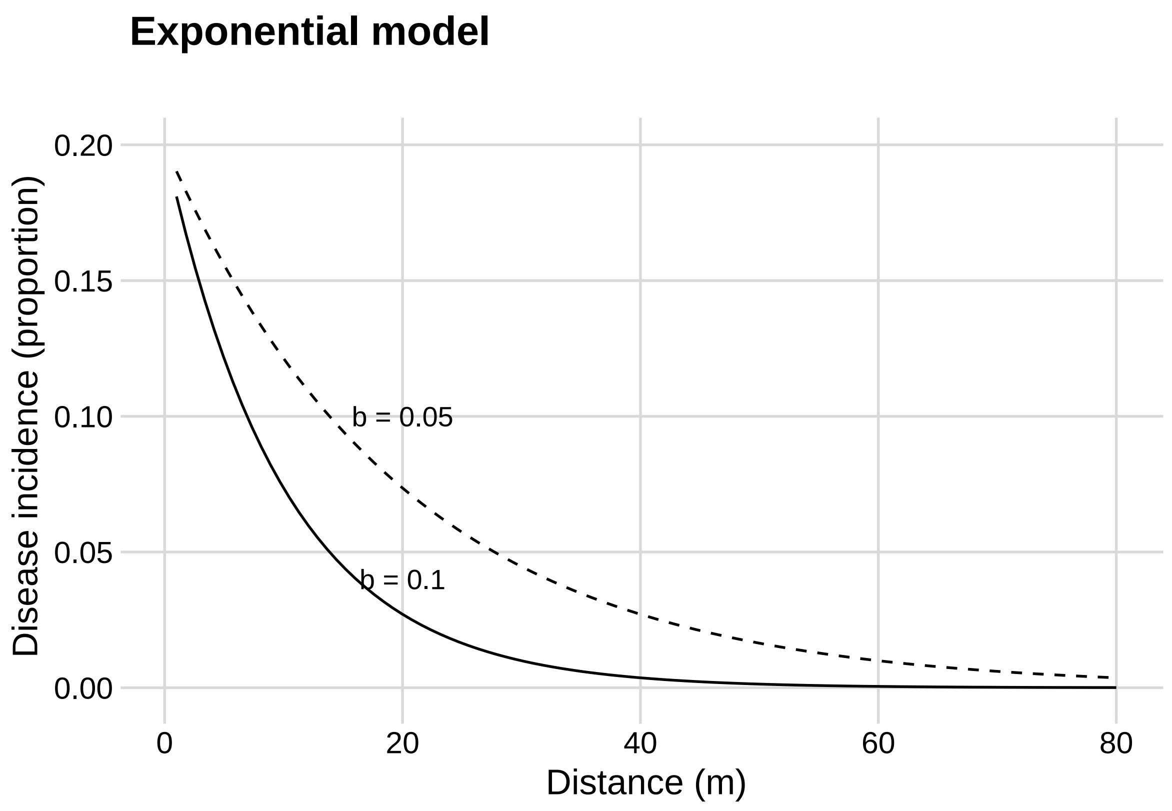
```r
library(tidyverse)
library(ggthemes)
library(patchwork)
library(cowplot) # for themes
theme_set(theme_minimal_grid()) # set global theme
```

Set the parameters for the exponential model with two rates and same inoculum at the source:

```r
a1 <- 0.2 # y at distance zero for gradient 1
a2 <- 0.2 # y at distance zero for gradient 2
b1 <- 0.1 # decline rate for gradient 1
b2 <- 0.05 # decline rate for gradient 2
max1 <- 80 # maximum distance for gradient 1
max2 <- 80 # maximum distance for gradient 2
dat <- data.frame(x = seq(1:max1), y = seq(0:a1))
```

The following code allows to visualize the model predictions.

```r
dat %>%
  ggplot(aes(x, y)) +
  stat_function(fun = function(x) a1 * exp(-b1 * x), linetype = 1) +
  stat_function(fun = function(x) a2 * exp(-b2 * x), linetype = 2) +
  ylim(0, a1) +
  annotate("text", x = 20, y = 0.04, label = "b = 0.1") +
  annotate("text", x = 20, y = 0.10, label = "b = 0.05") +
  labs(
    title = "Exponential model",
    subtitle = "",
    x = "Distance (m)",
    y = "Disease incidence (proportion)"
  )
```



2.1.2 Power law model

Also known as the modified Gregory's model (Gregory was a pioneer in the use this model to describe plant disease gradients). In the power law model, Y is proportional to the power of the distance, and is given by:

$Y = a_P.x - b_P$

where a_P and b_P are the two parameters of the power law model. They differ from the exponential because as closer to x is to zero, Y is indefinitely large (not meaningful biologically). However, the model can still be useful because it produces realistic values at any distance x away from the source. The values of the a_P parameter should be interpreted in accord to the scale of x, whether in centimeters or meters. If the distance between the source and the first measure away from the source is 0.5m, it is so more appropriate to record the distance in cm than in m or km.

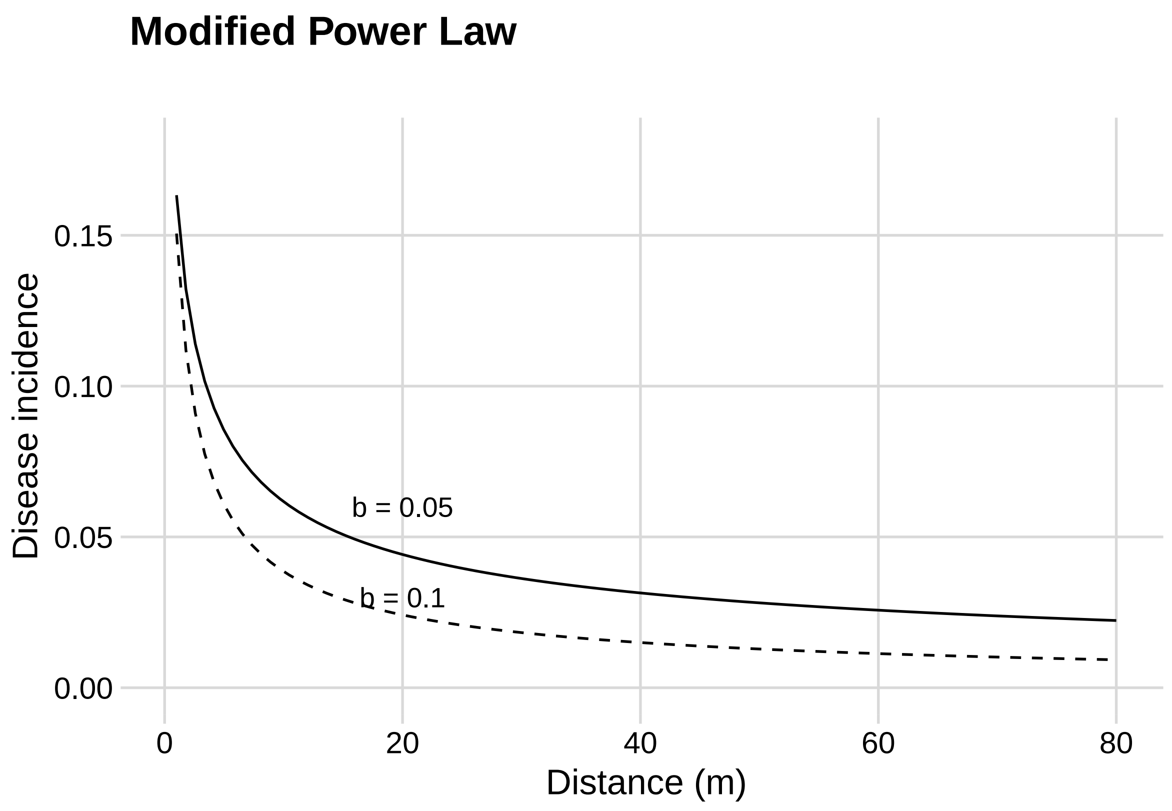
Once y at the distance zero from the source is undefined when using the power law model, this is usually modified by the addition of a positive constant C in x:

$Y = a_P.(x + C) - b_P$

For this reason, the model is named as the modified power law. Here, the constant C is of the same unit of x. At the distance zero, the positive constant is a term that express the size of the inoculum source. In other words, the a parameter is a theoretical value of Y at the distance $1 - C$ from the center of the inoculum source.

Let's plot two gradients with two rate parameters for the modified power law model:

```
C <- 0.5
a1 <- 0.2 # y at zero distance for gradient 1
a2 <- 0.2 # y at zero distance for gradient 2
b1 <- 0.5 # decline rate for gradient 1
b2 <- 0.7 # decline rate for gradient 2
max1 <- 80 # maximum distance for gradient 1
max2 <- 80 # maximum distance for gradient 2
dat2 <- data.frame(x = seq(1:max1), y = seq(0:a1))


dat2 %>%
  ggplot(aes(x, y)) +
  stat_function(fun = function(x) a1 * ((x + C)^-b1), linetype = 1) +
  stat_function(fun = function(x) a2 * ((x + C)^-b2), linetype = 2) +
  ylim(0, a1 - 0.02) +
  annotate("text", x = 20, y = 0.03, label = "b = 0.1") +
  annotate("text", x = 20, y = 0.06, label = "b = 0.05") +
  labs(
    title = "Modified Power Law",
    subtitle = "",
    x = "Distance (m)",
    y = "Disease incidence"
  )
```



The differential equation of the power law model is given by:

$\frac{dy}{dx} = \frac{-b_P.Y}{x-C}$

Similar to the exponential model, $\frac{dy}{dx}$ is proportional to Y, meaning that the gradient is steeper (more negative) at the highest disease intensity value, usually closer to the source.

2.2 Linearization of the models

2.2.1 Transformations of y

The gradient models, again similar to the temporal disease models, are **non linear in their parameters**. The model is intrinsically linear if transformations are applied (according to the model) in both sides of the equations. The linear model in its generic state is given by

$y* = a * +bx$,

where the asterisk in a indicated that one of the transformations was applied in y that produced the linear model. Note that $a*$ is the transformed version of the initial disease intensity, which needs to be returned to the original scale according to the respective back-transformation. Follows the linearized form of the two most common gradient models.

$ln(y) = ln(a_E) - b_E.x$

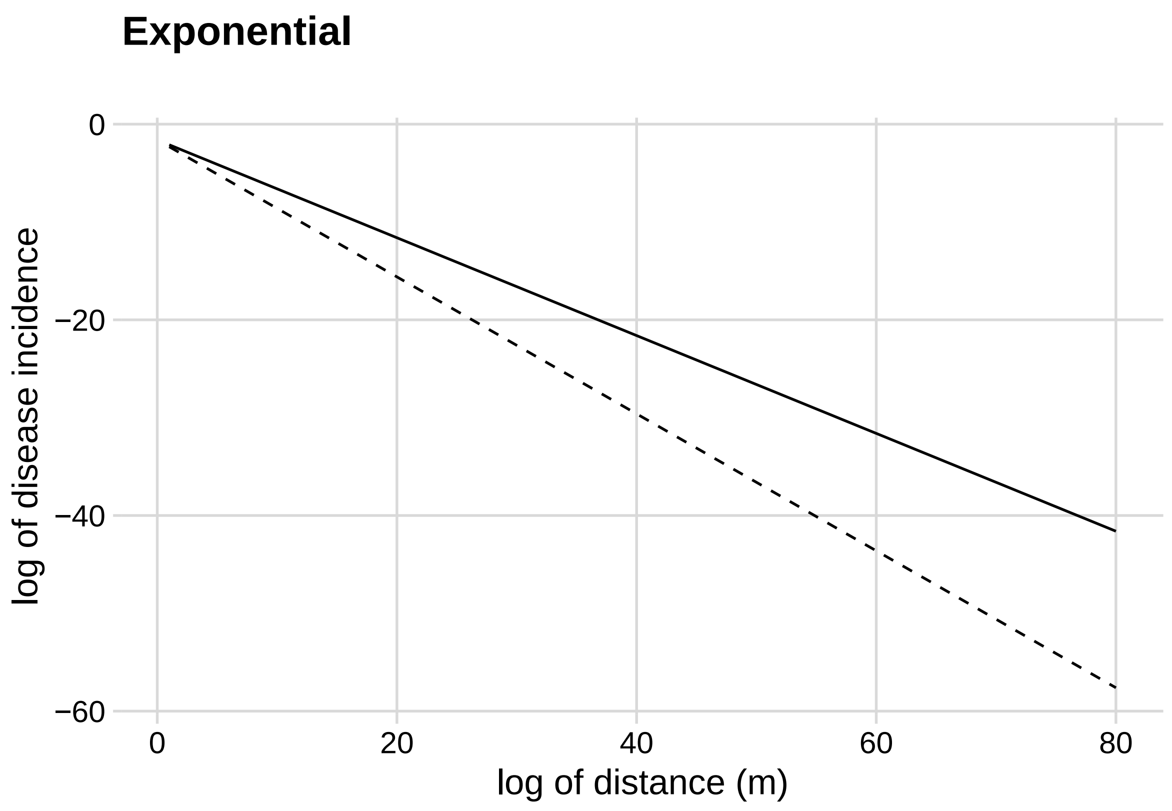
$ln(y) = ln(a_P) - b_E.ln(x + C)$

2.2.2 Plot for the linearized form of models

Let's visualize the linearization of the exponential model with two different slopes (gradient 1 and 2). Note that the transformation used was $ln(y)$.

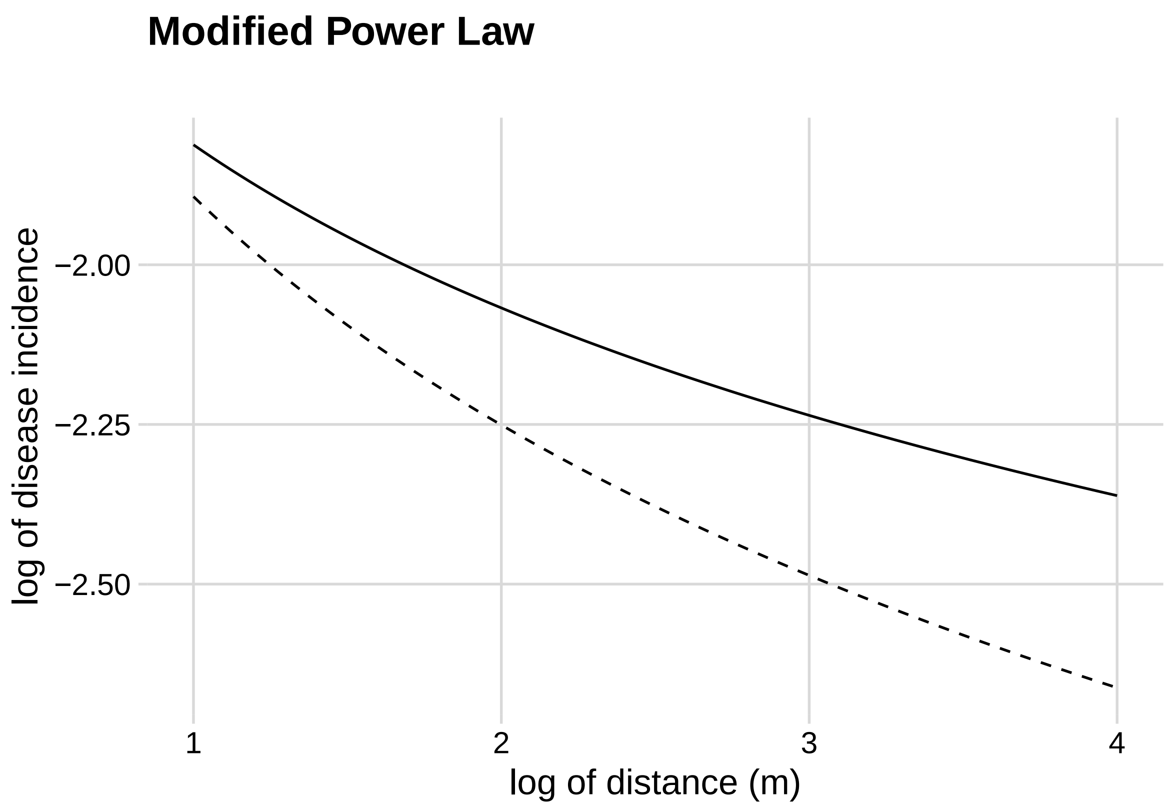
Follows the linearization of the modified power law model.

```
C <- 0.5
a1 <- 0.2 # y at zero distance for gradient 1
a2 <- 0.2 # y at zero distance for gradient 2
b1 <- 0.5 # decline rate for gradient 1
b2 <- 0.7 # decline rate for gradient 2
max1 <- 80 # maximum distance for gradient 1
max2 <- 80 # maximum distance for gradient 2
dat2 <- data.frame(x = seq(1:max1), y = seq(0:a1))

dat2 %>%
  ggplot(aes(x, y)) +
  stat_function(fun = function(x) log(a1) - (b1 * x), linetype = 1) +
  stat_function(fun = function(x) log(a2) - (b2 * x), linetype = 2) +
  labs(
    title = "Exponential",
    subtitle = "",
    x = "log of distance (m)",
    y = "log of disease incidence"
  )
```



Follows the linearization of the modified power law model. Note that the transformation used was $ln(y)$ and $ln(x + C)$.

```r
C <- 0.5
a1 <- 0.2 # y at zero distance for gradient 1
a2 <- 0.2 # y at zero distance for gradient 2
b1 <- 0.5 # decline rate for gradient 1
b2 <- 0.7 # decline rate for gradient 2
max1 <- log(80) # maximum distance for gradient 1
max2 <- log(80) # maximum distance for gradient 2
dat2 <- data.frame(x = seq(1:max1), y = seq(0:a1))

dat2 %>%
  ggplot(aes(x, y)) +
  stat_function(fun = function(x) log(a1) - (b1 * log(x + C)), linetype = 1) +
  stat_function(fun = function(x) log(a2) - (b2 * log(x + C)), linetype = 2) +
  labs(
    title = "Modified Power Law",
    subtitle = "",
    x = "log of distance (m)",
    y = "log of disease incidence"
  )
```



2.3 Model fitting

2.3.1 Dataset

The hypothetical data below shows a gradient for the number of lesions counted at varying distances in meters from the source. Let's create two vectors, one for the distances x and the other for the lesion count Y, and then a dataframe by combining the two vectors.

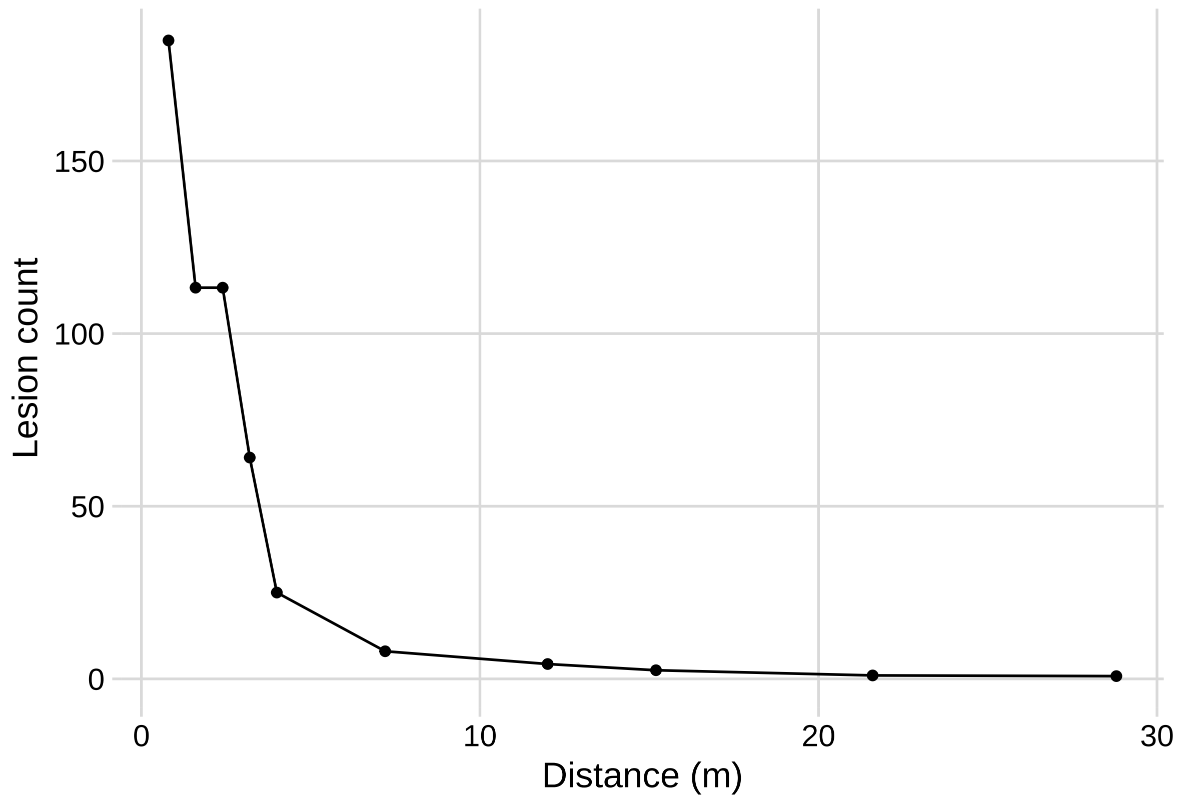
```
# create the two vectors
x <- c(0.8, 1.6, 2.4, 3.2, 4, 7.2, 12, 15.2, 21.6, 28.8)
Y <- c(184.9, 113.3, 113.3, 64.1, 25, 8, 4.3, 2.5, 1, 0.8)
grad1 <- data.frame(x, Y) # create the dataframe
grad1 # show the gradient
```

```
     x     Y
1  0.8 184.9
```

```
2   1.6 113.3
3   2.4 113.3
4   3.2  64.1
5   4.0  25.0
6   7.2   8.0
7  12.0   4.3
8  15.2   2.5
9  21.6   1.0
10 28.8   0.8
```

2.3.2 Visualize the gradient

```
grad1 %>%
  ggplot(aes(x, Y))+
  geom_point()+
  geom_line()+
  labs(y = "Lesion count",
       x = "Distance (m)")
```



2.3.3 Linear regression

A linear regression model is fitted to the transformed variables according to the model. The higher the coefficient of determination, the better is the fit of the model to the data.

Exponential model

```
reg_exp <- lm(log(Y) ~ x, data = grad1)
summary(reg_exp)
```

```

Call:
lm(formula = log(Y) ~ x, data = grad1)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.04868 -0.58973 -0.00144  0.59572  0.99554 
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  4.57705    0.35222  12.995 1.17e-06 ***
x           -0.20124    0.02656  -7.576 6.45e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7612 on 8 degrees of freedom
Multiple R-squared:  0.8777,	Adjusted R-squared:  0.8624 
F-statistic: 57.39 on 1 and 8 DF,  p-value: 6.45e-05
```

Power law model with $C = 0$.

```r
reg_p <- lm(log(Y) ~ log(x), data = grad1)
summary(reg_p)
```

```

Call:
lm(formula = log(Y) ~ log(x), data = grad1)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.72281 -0.11989 -0.03146  0.08755  0.65267 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   5.5638     0.2456   22.66 1.53e-08 ***
log(x)       -1.6978     0.1191  -14.26 5.71e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4235 on 8 degrees of freedom
Multiple R-squared:  0.9621,	Adjusted R-squared:  0.9574 
F-statistic: 203.3 on 1 and 8 DF,  p-value: 5.71e-07
```

Power law model with $C = 0.4$.

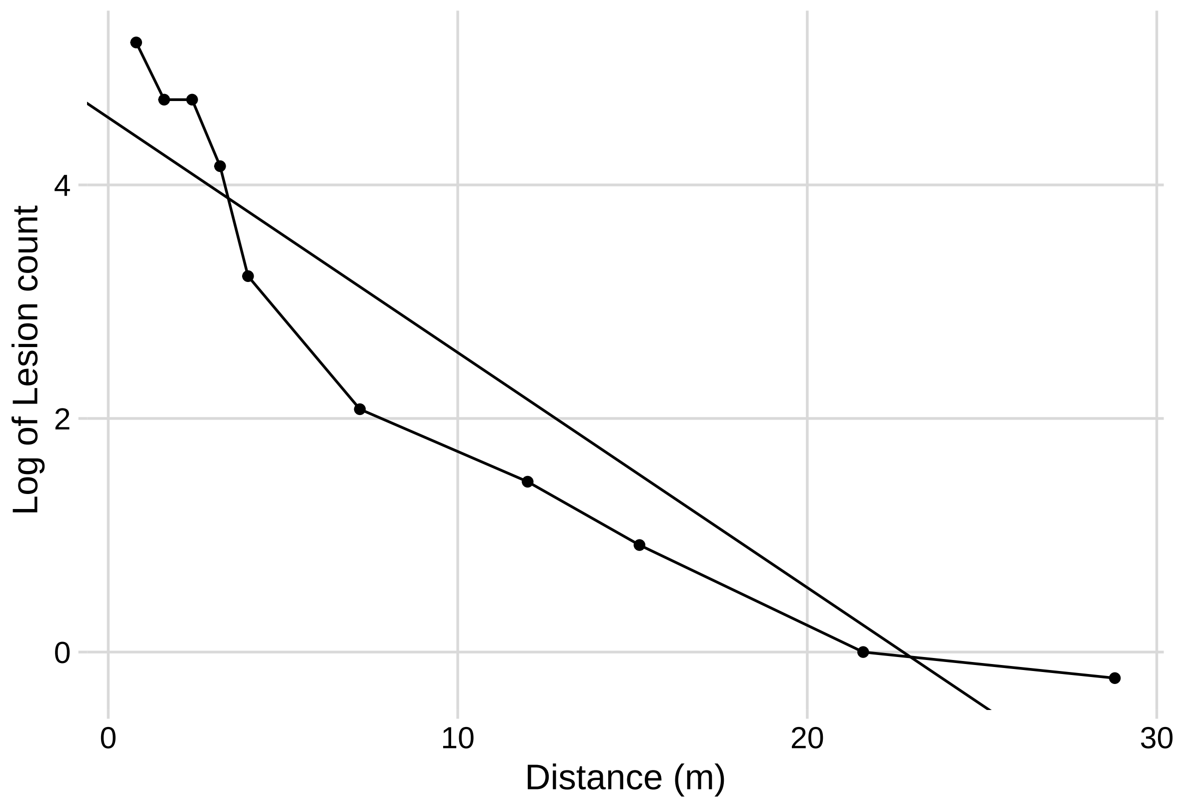
```r
reg_pm <- lm(log(Y) ~ log(x + 0.4), data = grad1)
summary(reg_pm)
```

```

Call:
lm(formula = log(Y) ~ log(x + 0.4), data = grad1)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.53733 -0.17258 -0.03646  0.08450  0.56928 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)    6.1007     0.2283   26.73 4.13e-09 ***
log(x + 0.4)  -1.8841     0.1084  -17.38 1.22e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3495 on 8 degrees of freedom
Multiple R-squared:  0.9742,	Adjusted R-squared:  0.971 
F-statistic: 302.2 on 1 and 8 DF,  p-value: 1.223e-07
```

Graphs for the fitted models

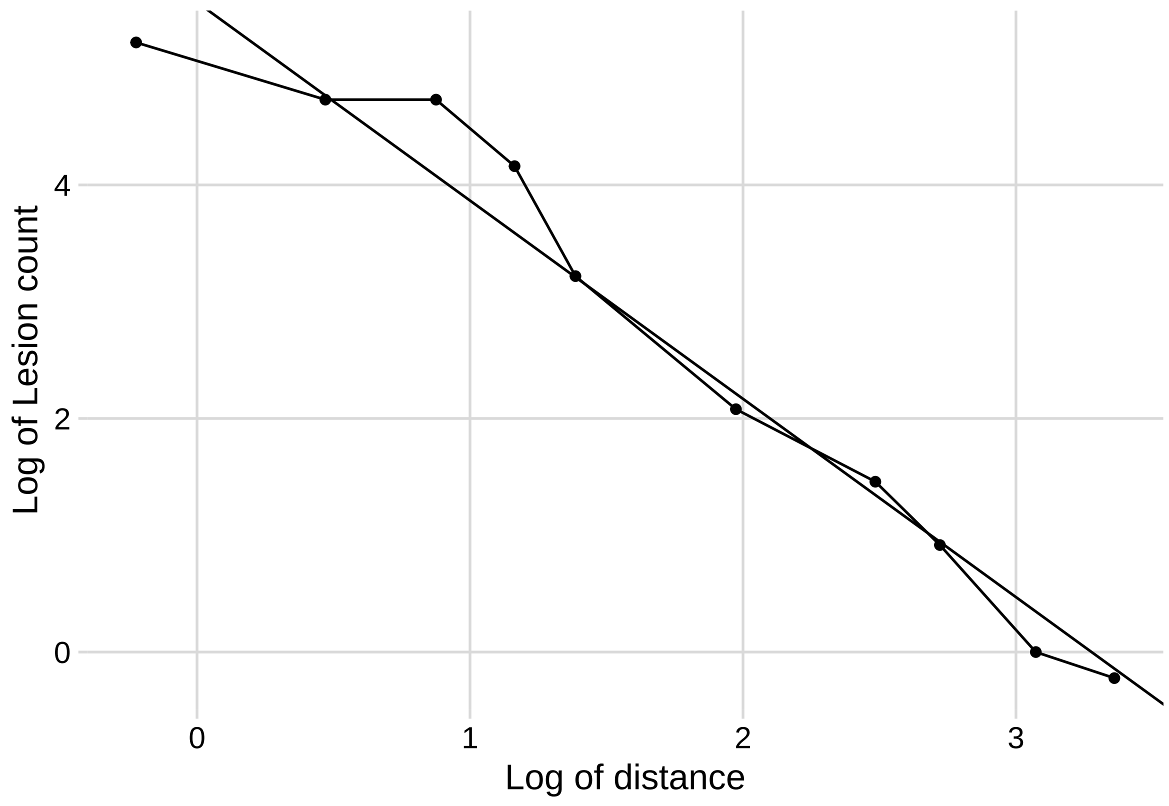
Exponential

```r
grad1 %>%
  ggplot(aes(x, log(Y)))+
  geom_point()+
  geom_line()+
  geom_abline(slope = coef(reg_exp)[[2]], intercept = coef(reg_exp)[[1]])+
 labs(y = "Log of Lesion count",
       x = "Distance (m)")
```



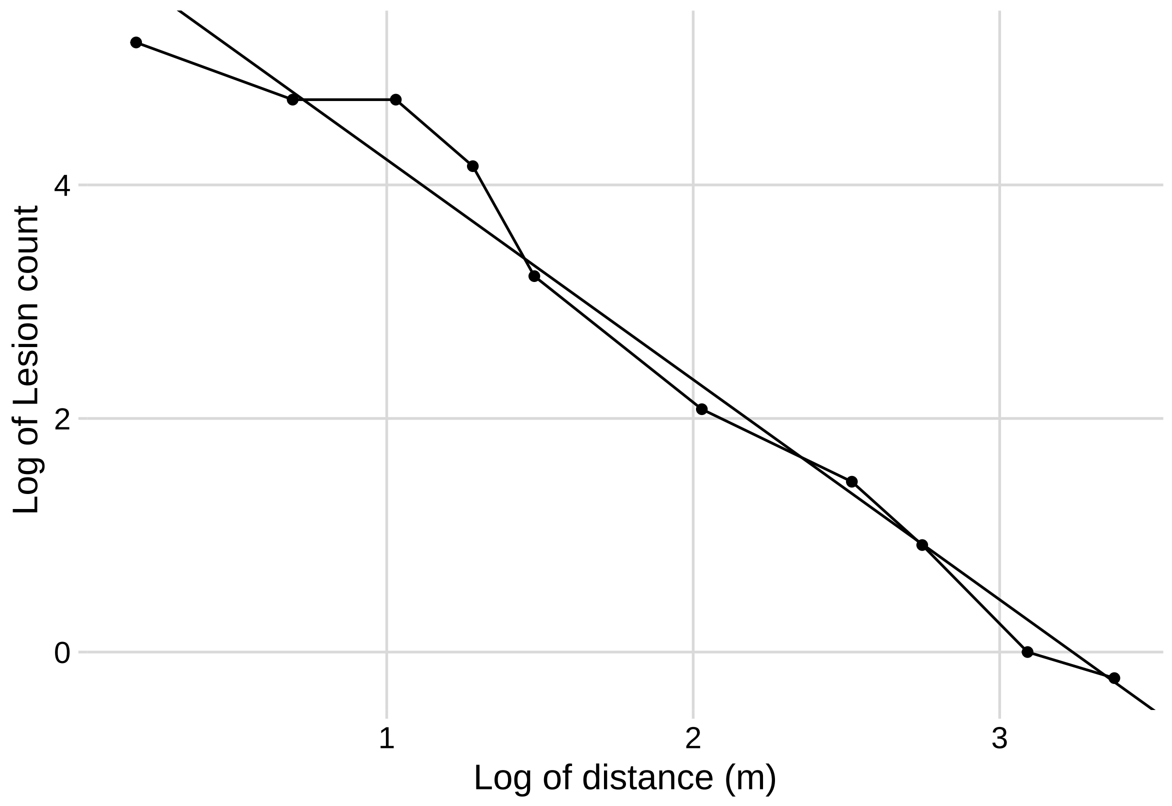
Power law model

```
grad1 %>%
  ggplot(aes(log(x), log(Y)))+
  geom_point()+
  geom_line()+
  geom_abline(slope = coef(reg_p)[[2]], intercept = coef(reg_p)[[1]])+
 labs(y = "Log of Lesion count",
       x = "Log of distance")
```



Modified power law model

```
grad1 %>%
  ggplot(aes(log(x+0.4), log(Y)))+
  geom_point()+
  geom_line()+
  geom_abline(slope = coef(reg_pm)[[2]], intercept = coef(reg_pm)[[1]])+
 labs(y = "Log of Lesion count",
       x = "Log of distance (m)")
```



Conclusion: The modified power law model provided the best fit.

3 Spatial patterns

3.1 Introduction

A spatial disease pattern can be defined as the arrangement of diseased entities relative to each other and to the architecture of the host crop (Madden et al. 2017c) . Such arrangement is the realization of the underlying dispersal of the pathogen, from one or several sources within and/or outside the area of interest, under the influence of physical, biological and environmental factors.

The study of spatial patterns is conducted at a specific time or multiple times during the epidemic. When assessed multiple times, both spatial and temporal processes can be characterized. Because epidemics change over time, it is expected that spatial patterns are not constant but change over time as well. Usually, plant pathologists are interested in determining spatial patterns at one or various spatial scales, depending on the objective of the study. The scale of interest may be a leaf or root, plant, field, municipality, state, country or even intercontinental area. The diseased units observed may vary from lesions on a single leaf to diseased fields in a large production region.

The patterns can be classified into two main types that occur naturally: **random** or **aggregated**. The random pattern originates because the chances for the units (leaf, plant, crop) to be infected are equal and low, and are largely independent from each other. In aggregated spatial patterns, such chances are unequal and there is dependency among the units, for example, a healthy unit close to a diseased unit is at higher risk than more distant units.

A range of techniques, most based on statistical tests, can be used to detect deviations from randomness in space and the choice of the methods depends on the scale of observation. Usually, more than one test is applied for the same or different scales of interest depending on how the data are collected. The several statistical tests can be classified based on the spatial scale and type of data (binary, count, etc) collected, but mainly if the spatial location of the unit is known (mapped) or not known (sampled). Following Madden et al. (2007), two major groups can be formed. The sparsely sampled (incidence or count data) data or intensively mapped (binary or grouped data) data.

3.2 Intensively mapped

3.2.1 Binary data

In this situation the individual plants are mapped, meaning that their relative positions to one another are known. It is the case when a census is used to map presence/absence data. The status of each unit (usually a plant) is noted is a binay variable. The plant is either diseased (D or 1) or non-diseased or healthy (H or 0). Several statistical tests can be used to detect a deviation from randomness. The most commonly used tests are runs, doublets and join count.

3.2.1.1 Runs test

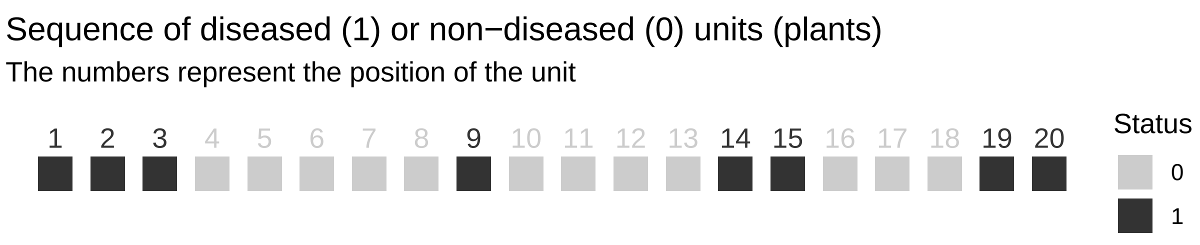
A run is defined as a succession of one or more diseased or healthy plants, which are followed and preceded by a plant of the other disease status or no plant at all. There would be few runs if there is an aggregation of diseased or healthy plants and a large number of runs for a random mixing of diseased and healthy plants.

Let's create a vector of binary (0 = non-diseased; 1 = diseased) data representing a crop row with 20 plants and assign it to y. For plotting purposes, we make a dataframe for more complete information.

```
y1 <- c(1,1,1,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,1)
x1 <- c(1:20) # position of each plant
z1 <- 1
row1 <- data.frame(x1, y1, z1) # create a dataframe
```

We can then visualize the series using ggplot and count the number of runs as 7, aided by the color used to identify a run.

```
library(tidyverse)
row1 %>%
  ggplot(aes(x1, z1, label = x1, color = factor(y1)))+
  geom_point(shape =15, size =6)+
  theme_void()+
  scale_x_continuous(breaks = max(z1))+
  scale_color_manual(values = c("grey80", "grey20"))+
  geom_text(vjust = 0, nudge_y = 0.5)+
coord_fixed()+
  ylim(-0.5,2.5)+
  theme(legend.position = "right")+
  labs(color = "Status", title = "Sequence of diseased (1) or non-diseased (0) units (plan
       subtitle = "The numbers represent the position of the unit")
```



We can write a code in R and create a function named `oruns.test` for the ordinary runs test.

```r
oruns.test <- function(x) {
# identify the sequence
S <- x
# Compute the number or runs
U = max(cumsum(c(1, diff(S)!=0)))
# Compute the number of diseased plants
m = sum(S)
# Count the total number of plants
N = length(S)
# Calculate the number of expected runs
EU = 1 + (2 * m*(N - m)/N)
# Calculate the standard deviation in the sample
sU = sqrt(2 * m * (N - m) * (2 * m *(N-m)-N)/ (N^2 *(N-1)))
# Calculate the z-value
Z = (U - EU)/sU
# Obtain the p-value for the Z
pvalue <- (2*pnorm(abs(Z), lower.tail=FALSE))
# test if Z is lower than 1.64
result <- ifelse(Z < 1.64,
c("clustering"),
c("randomness"))
# Print the results
print(paste("There are",U,"runs. The number of expected runs is", round(EU,1), "P-value:",
}
```

We can now run the test for the example series above.

```r
oruns.test(row1$y1)
```

```
[1] "There are 7 runs. The number of expected runs is 10.6 P-value: 0.084166 . Alternative hy
```

There are built-in functions in R packages that allow for running the ordinary runs test. Let's load the packages and runt the test. Note that the results of the `runs.test` is the same as the one produced by our custom function.

```r
library(randtests)
runs.test(row1$y1, threshold = 0.5)
```

```

    Runs Test
```

```
data:  row1$y1
statistic = -1.727, runs = 7, n1 = 8, n2 = 12, n = 20, p-value =
0.08417
alternative hypothesis: nonrandomness
```

```r
library(DescTools)
r <- RunsTest(row1$y1)
```

3.2.1.2 Doublets

Doublet analysis is used to compare the observed number or adjacent diseased plants, a doublet (DD or 11), to the number expected if the disease were randomly distributed in the yard. If the observed number is greater than the expected number, contagion within the field is suspected.

Let's manually produce a code to execute the doublets test. To facilitate, we can create a function and name it `doublets.test`. The only argument needed is the vector of binary data.

```r
doublets.test <- function(x) {
# Identify the sequence
S <- x
# Compute the number of doublets Db
matrix <- cbind(S[-length(S)], S[-1])
pairs <- table(data.frame(matrix))
Db <- pairs[2,2]
# Count the number of diseased plants
N <- length(S)
# Count the number of total plants
m = sum(S)
# Expected number of doublets
EDb = m *((m -1)/N)
# Standard deviation
SDb = sqrt ( EDb * (1 - (2 / N)))
# Calculate the Z-value
ZDb = (Db - EDb)/ SDb
# two-sided P-value calculation
pvalue <- (2*pnorm(abs(ZDb), lower.tail = FALSE))
# Result of the test
result <- ifelse(abs(ZDb) >= 1.64,
c("aggregation or clustering"),
c("randomness"))
# Print the results
print(paste("There are",Db,"doublets. The number of expected doublets is",EDb,".","P-value
}
```

```r
# Run the function calling the vector
doublets.test(row1$y1)
```

```
[1] "There are 4 doublets. The number of expected doublets is 2.8 . P-value: 0.4497 . Altern
```

3.2.1.3 Join count

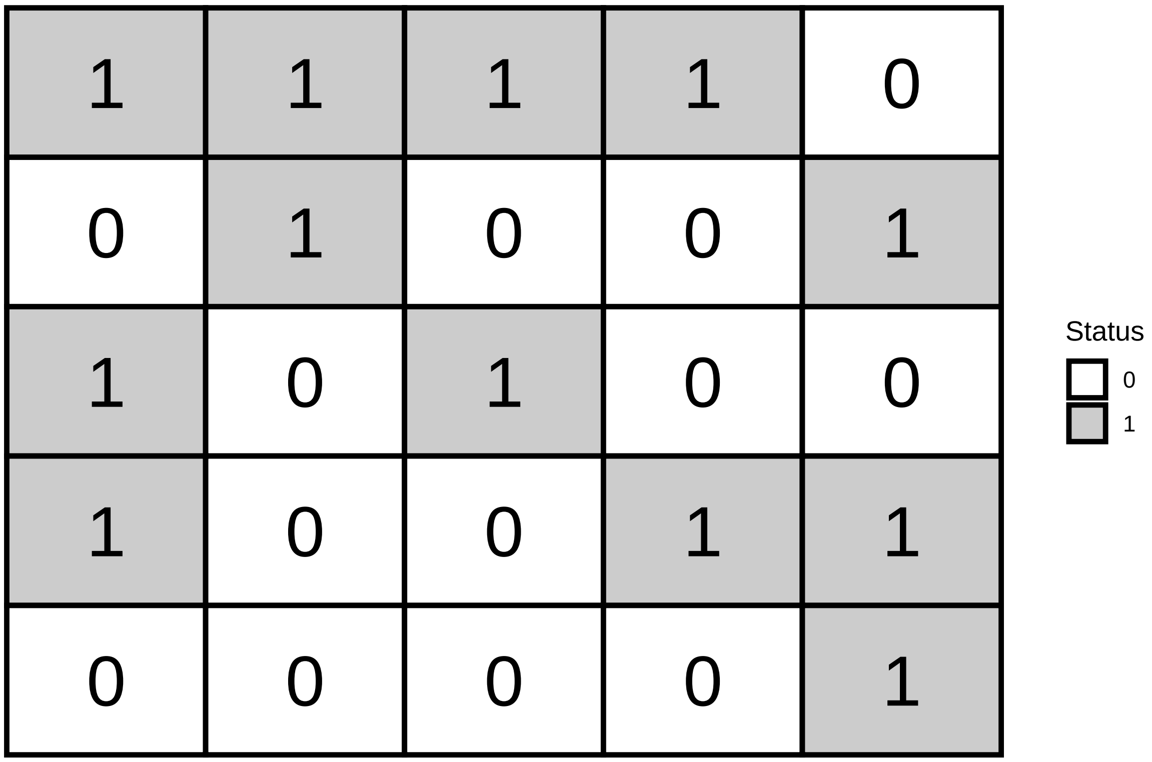
In this analysis, two adjacent plants may be classified by the type of join that links them: D-D, H-H or H-D. The orientation(s) of interest (along rows, across rows, diagonally, or a a combination o these) should be specified. The number of joins of the specified type in the orientation(s) of interest is then counted. The question is whether the observed join-count is large (or small) relative to that expected for a random pattern. The join-count statistics provides a basic measure of spatial autocorrelation.

In R, we can use the `join.count` function of the `spdep` package to perform a joint count test. First we need to create the series of binary data from top to bottom and left to right. The data are shown in Fig. 9.13 in page 260 of the book chapter on spatial analysis (Madden et al. 2017c). In the example, there are 5 rows and 5 columns. This will be informed later to run the test.

```r
# Enter the data
S2 <- c(1,0,1,1,0,
       1,1,0,0,0,
       1,0,1,0,0,
       1,0,0,1,0,
       0,1,0,1,1)
```

Visualize the two-dimensional array:

```r
# Convert to raster
mapS2 <- raster::raster(matrix(S2, 5 ,5))

# Convert to data frame
mapS3 <- raster::as.data.frame(mapS2,xy=TRUE)

# Map using ggplot
mapS3 %>%
  ggplot(aes(x, y, label = layer, fill = factor(layer)))+
  geom_tile(color = "black", size =1)+
  theme_void()+
  geom_text(size = 10)+
  labs(fill = "Status")+
  scale_fill_manual(values = c("white", "grey80"))
```



Load the library

```
library(spdep)
```

First, we need to generate a list of neighbors (nb) for a grid of cells. This is performed with the `cell2nb` function by informing the number of rows and columns. The argument "rook" means shared edge, but it could be the "queen", for shared edge or vertex. We can use the default.

```
nb <- cell2nb(nrow = 5,
              ncol = 5,
              type="rook")
```

The `joincount.test` function runs the BB join count test for spatial autocorrelation. From the function description, the method uses a spatial weights matrix in weights list form for testing whether same-status joins occur more frequently than would be expected if the zones were labelled in a spatially random way. We need to inform the sequence as factor and the nb object we created previously.

```
joincount.test(factor(S2),
                nb2listw(nb))
```

```

    Join count test under nonfree sampling

data:  factor(S2)
weights: nb2listw(nb)

Std. deviate for 0 = -0.58266, p-value = 0.7199
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation              Variance
            2.9583333             3.2500000             0.2505797


    Join count test under nonfree sampling

data:  factor(S2)
weights: nb2listw(nb)

Std. deviate for 1 = -0.66841, p-value = 0.7481
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation              Variance
            2.4166667             2.7500000             0.2486957
```

The function returns a list with a class for each of the status (in this case 0 and 1) with several components. We should look at the **P-value**. The alternative hypothesis (greater) is that the same status joins occur more frequently than expected if they were labelled in a spatial random way. In this case, we do not reject the null hypothesis of randomness.

We can run the ordinary runs and doublets tests, which only considers the adjacent neighbor, for the same series and compare the results.

```
oruns.test(S2)
```

```
[1] "There are 17 runs. The number of expected runs is 13.5 P-value: 0.149673 . Alternative 
```

```
doublets.test(S2)
```

```
[1] "There are 3 doublets. The number of expected doublets is 5.28 . P-value: 0.3009 . Alter
```

Let’s repeat the procedure using the second array of data shown in the book chapter, for which the result is different. In this case, there is evidence to reject the null hypothesis, indicating aggregation of plants.

```
S3 <- c(1,1,1,0,0,
       1,1,1,0,0,
       1,1,1,0,0,
       1,1,1,0,0,
       0,0,0,0,0)

joincount.test(factor(S3),
                nb2listw(nb))
```

```

    Join count test under nonfree sampling

data:  factor(S3)
weights: nb2listw(nb)

Std. deviate for 0 = 4.2451, p-value = 1.093e-05
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation              Variance
            5.3750000             3.2500000             0.2505797


    Join count test under nonfree sampling

data:  factor(S3)
weights: nb2listw(nb)

Std. deviate for 1 = 4.5953, p-value = 2.16e-06
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation              Variance
            5.0416667             2.7500000             0.2486957
```

```
oruns.test(S3)
```

```
[1] "There are 8 runs. The number of expected runs is 13.5 P-value: 0.024904 . Alternative hy
```

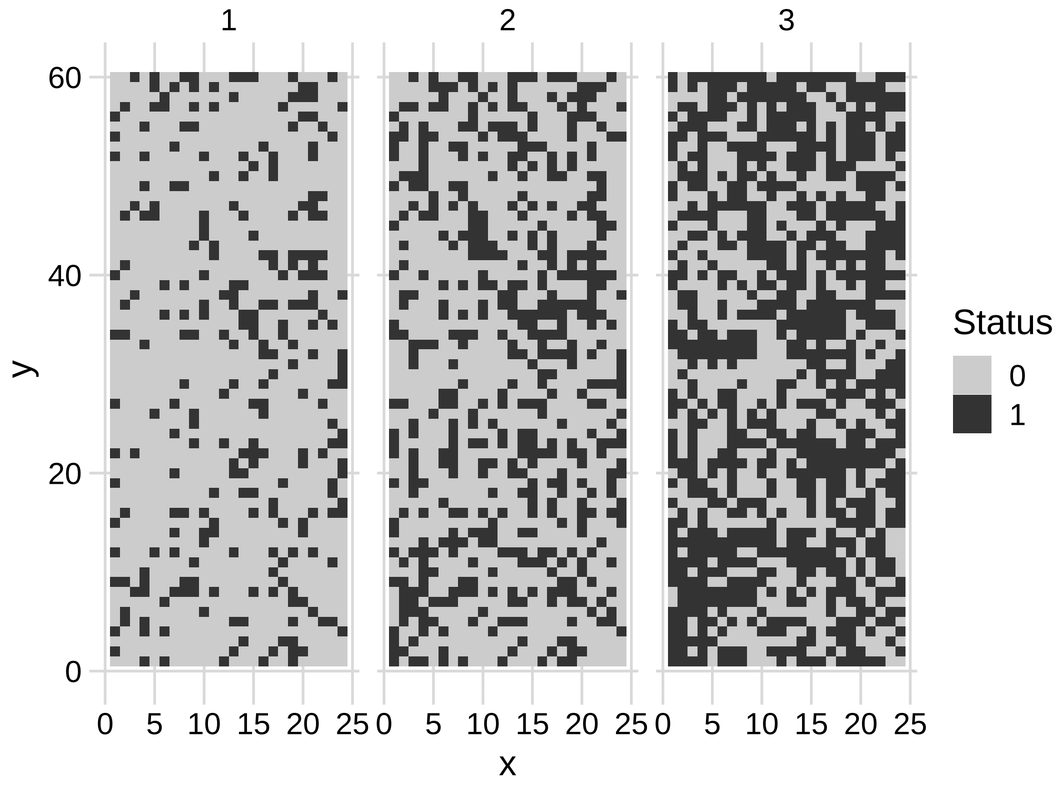
We can apply these tests for a real example epidemic data provided by the epiphy R package. Let's work with part of the intensively mapped data on the incidence of tomato spotted wilt virus (TSWV) disease in field trials reported by Cochran (1936) and Bald (1937). First, we need to load the library and then assign one dataframe (the dataset has two dataframes) of the dataset `tomato_tswv` to a new dataframe called `tswv_1929`.

```r
library(epiphy)
library(cowplot) # theming the ggplot
tswv_1929 <- tomato_tswv$field_1929
head(tswv_1929)
```

```
  x y t i n
1 1 1 1 0 1
2 1 2 1 1 1
3 1 3 1 0 1
4 1 4 1 1 1
5 1 5 1 0 1
6 1 6 1 0 1
```

The inspection of the first rows of the dataframe shows five variables where x and y are spatial grid coordinates, t is assessment time, i is the status of the plant (0 = healthy, 1 = diseased) and n is the sampling unit size (here all one). Let's visualize these data for each sampling time.

```r
tswv_1929 %>%
  ggplot(aes(x, y, fill = factor(i)))+
  geom_tile()+
  coord_fixed()+
  theme_minimal_grid()+
  scale_fill_grey(start = 0.8, end = 0.2)+
  facet_wrap(~ t)+
  labs(fill = "Status")
```



Check the number of rows (y) and columns (x) for further preparing the neighbor object for the join count statistics.

```
tswv_1929 %>%
  select(x, y) %>%
  summary()
```

```
       x               y        
 Min.   : 1.00   Min.   : 1.00  
 1st Qu.: 6.75   1st Qu.:15.75  
 Median :12.50   Median :30.50  
 Mean   :12.50   Mean   :30.50  
 3rd Qu.:18.25   3rd Qu.:45.25  
 Max.   :24.00   Max.   :60.00  
```

There are 60 rows and 24 columns.

```r
# Neighbor grid
nb1 <- cell2nb(nrow = 60,
              ncol = 24,
              type="rook")

# Pull the binary sequence of time 1
S1 <- tswv_1929 %>%
  filter(t == "1") %>%
  pull(i)

joincount.test(factor(S1),
                nb2listw(nb1))
```

```

    Join count test under nonfree sampling

data:  factor(S1)
weights: nb2listw(nb1)

Std. deviate for 0 = -0.28351, p-value = 0.6116
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation              Variance
           482.000000            482.578874              4.169132


    Join count test under nonfree sampling

data:  factor(S1)
weights: nb2listw(nb1)

Std. deviate for 1 = -0.059497, p-value = 0.5237
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation              Variance
            23.458333             23.578874              4.104614
```

We can apply the join count test for time 2 and time 3. Results show that the pattern changes from random to aggregate over time.

```
# Pull the binary sequence of time 1
S2 <- tswv_1929 %>%
  filter(t == "2") %>%
  pull(i)

joincount.test(factor(S2),
                nb2listw(nb1))
```

```
    Join count test under nonfree sampling

data:  factor(S2)
weights: nb2listw(nb1)

Std. deviate for 0 = 0.35872, p-value = 0.3599
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation              Variance
           317.000000            315.900625              9.392312


    Join count test under nonfree sampling

data:  factor(S2)
weights: nb2listw(nb1)

Std. deviate for 1 = 0.34604, p-value = 0.3647
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation              Variance
            82.958333             81.900625              9.342754
```

```
# Pull the binary sequence of time 1
S3 <- tswv_1929 %>%
  filter(t == "3") %>%
  pull(i)

joincount.test(factor(S3),
                nb2listw(nb1))
```

```
    Join count test under nonfree sampling
```

```
data:  factor(S3)
weights: nb2listw(nb1)

Std. deviate for 0 = 1.8541, p-value = 0.03186
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation             Variance
            136.12500             129.92773             11.17243


    Join count test under nonfree sampling

data:  factor(S3)
weights: nb2listw(nb1)

Std. deviate for 1 = 1.7275, p-value = 0.04204
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation             Variance
            243.70833             237.92773             11.19743
```

3.2.2 Grouped data

If the data are intensively mapped, meaning that the spatial locations of the sampling units are known, we are not limited to analyse presence/absence (incidence) only data at the unit level. The sampling units may be quadrats where the total number of plants and the number of disease plants (or number of pathogen propagules) are known. Alternatively, it could be a continuous measure of severity. The question here, similar to the previous section, is whether a plant being diseased makes it more (or less) likely that neighboring plants will be diseased. If that is the case, diseased plants are exhibiting spatial autocorrelation. The most common methods are autocorrelation (known as Moran's I), semivariance and SADIE (an alternative approach to autocorrelation.)

3.2.2.1 Autocorrelation

Spatial autocorrelation analysis provides a quantitative assessment of whether a large value of disease intensity in a sampling unit makes it more (positive autocorrelation) or less (negative auto- correlation) likely that neighboring sampling units tend to have a large value of disease intensity (Madden et al. 2017c).

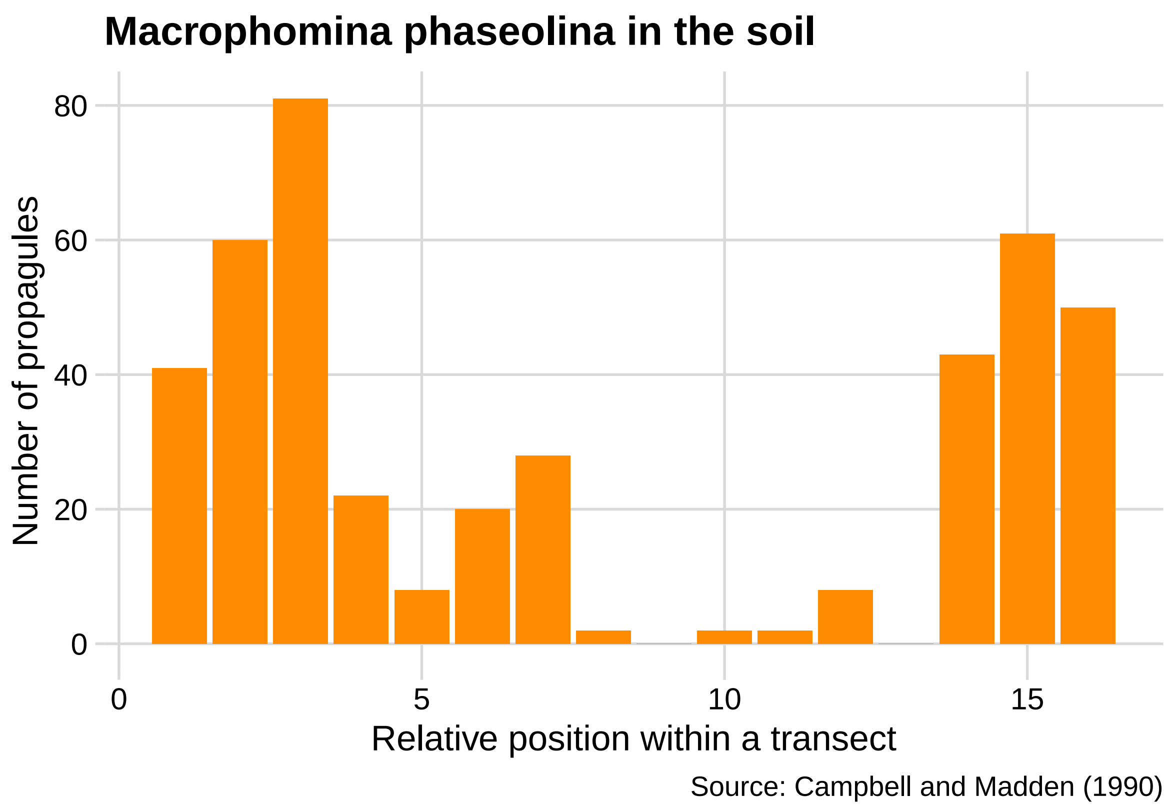
We will illustrate the method by reproducing the example provided in page 264 of the chapter on spatial analysis (Madden et al. 2017c), which was extracted from table 11.3 of Campbell and Madden (1990). The data represent a single transect with the number of *Macrophomia phaseolina* propagules per 10 g air-dry soil recorded in 16 contiguous quadrats across a field.

```
mp <- data.frame(
  i = c(1:16),
  y = c(41, 60, 81, 22, 8, 20, 28, 2, 0, 2, 2, 8, 0, 43, 61, 50)
)
mp
```

```
    i  y
1   1 41
2   2 60
3   3 81
4   4 22
5   5  8
6   6 20
7   7 28
8   8  2
9   9  0
10 10  2
11 11  2
12 12  8
13 13  0
14 14 43
15 15 61
16 16 50
```

We can produce a plot to visualize the number of propagules across the transect.

```
mp %>%
  ggplot(aes(i, y))+
  geom_col(fill = "darkorange")+
  theme_minimal_grid()+
  labs(x = "Relative position within a transect",
       y = "Number of propagules",
       title = "Macrophomina phaseolina in the soil",
       caption = "Source: Campbell and Madden (1990)")
```



To calculate the autocorrelation coefficient in R, we can use the `ac` function of the *tseries* package.

```
library(tseries)
ac_mp <- acf(mp$y, lag = 5, pl = FALSE)
ac_mp
```

```
Autocorrelations of series 'mp$y', by lag

     0      1      2      3      4      5 
 1.000  0.586  0.126 -0.033 -0.017 -0.181 
```

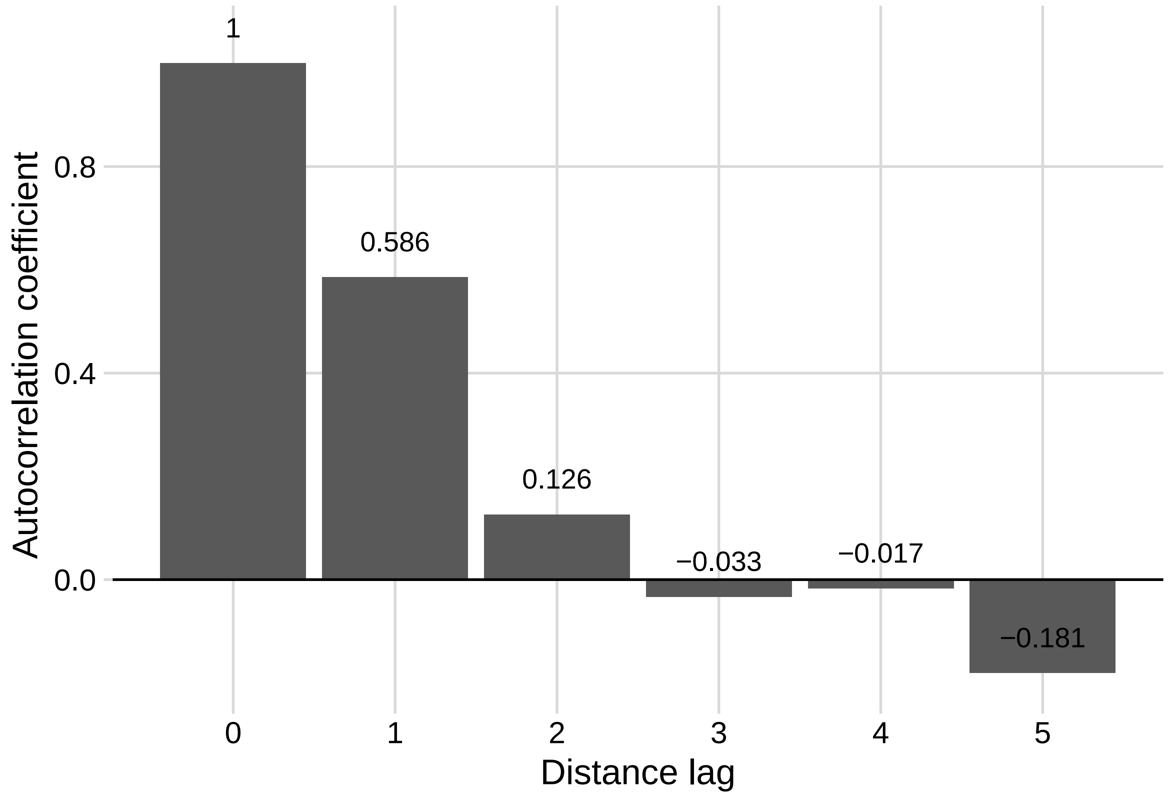
Let's store the results in a dataframe to facilitate visualization using ggplot.

```
ac_mp_dat <- data.frame(index = ac_mp$lag, ac_mp$acf)
ac_mp_dat
```

```
  index   ac_mp.acf
1     0  1.00000000
2     1  0.58579374
3     2  0.12636306
4     3 -0.03307249
5     4 -0.01701392
6     5 -0.18092810
```

And now the plot known as autocorrelogram.

```
ac_mp_dat %>%
  ggplot(aes(index, ac_mp.acf, label = round(ac_mp.acf,3)))+
  geom_col()+
  geom_text(vjust = 0, nudge_y = 0.05)+
  scale_x_continuous(n.breaks = 6)+
  theme_minimal_grid()+
  geom_hline(yintercept = 0)+
  labs(x = "Distance lag", y = "Autocorrelation coefficient")
```



The values we obtained here are not the same but quite close to the values reported in Madden et al. (2007). For the transect data, the calculated coefficients in the book example for lags 1, 2 and 3 are 0.625, 0.144, and - 0.041. The conclusion is the same, the smaller the distance between sampling units, the stronger is the correlation between the count values.

The method above is usually referred to Moran's I (Moran, 1950). Let's use another example dataset from the book to calculate the Moran's I in R. The data is shown in page 269 of the book. The data represent the number of diseased plants per quadrat (out of a total of 100 plants in each) in 144 quadrats. It was based on an epidemic generated using the stochastic simulator of Xu and Madden (2004). The data is stored in a csv file.

```
epi <- read_csv("data/xu-madden-simulated.csv")
# Transform from wide to long format
# Pull the n variable to store as a vector
epi1 <- epi %>%
  pivot_longer(2:13,
               names_to = "y",
               values_to = "n") %>%
  pull(n)
```

Using `moran` function of the *spdep* R package.

```
set.seed(100)
library(spdep)
```

The `cell2nb` function creates the neighbor list with 12 rows and 12 columns, which is how the 144 quadrats are arranged.

```
nb <- cell2nb(12, 12, type="queen", torus = FALSE)
```

The `nb2listw` function supplements a neighbors list with spatial weights for the chosen coding scheme. We use the default W, which is the row standardized (sums over all links to n). We then create the `col.W` neighbor list.

```
col.W <- nb2listw(nb, style="W")
```

The Moran's I statistic is given by the `moran` function

```r
moran(x = epi1, # numeric vector
      listw = col.W, # the nb list
      n = 12, # number of zones
      S0 = Szero(col.W)) # global sum of weights
```

```
$I
[1] 0.05818595

$K
[1] 2.878088
```

The Moran's test for spatial autocorrelation uses spatial weights matrix in weights list form.

```r
moran.test(x = epi1,
           listw = col.W)
```

```

	Moran I test under randomisation

data:  epi1  
weights: col.W    

Moran I statistic standard deviate = 15.919, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance 
      0.698231416      -0.006993007       0.001962596 
```

```r
correl_I <- sp.correlogram(nb, epi1,
                           order = 10,
                           method = "I",
                           zero.policy = TRUE)
```

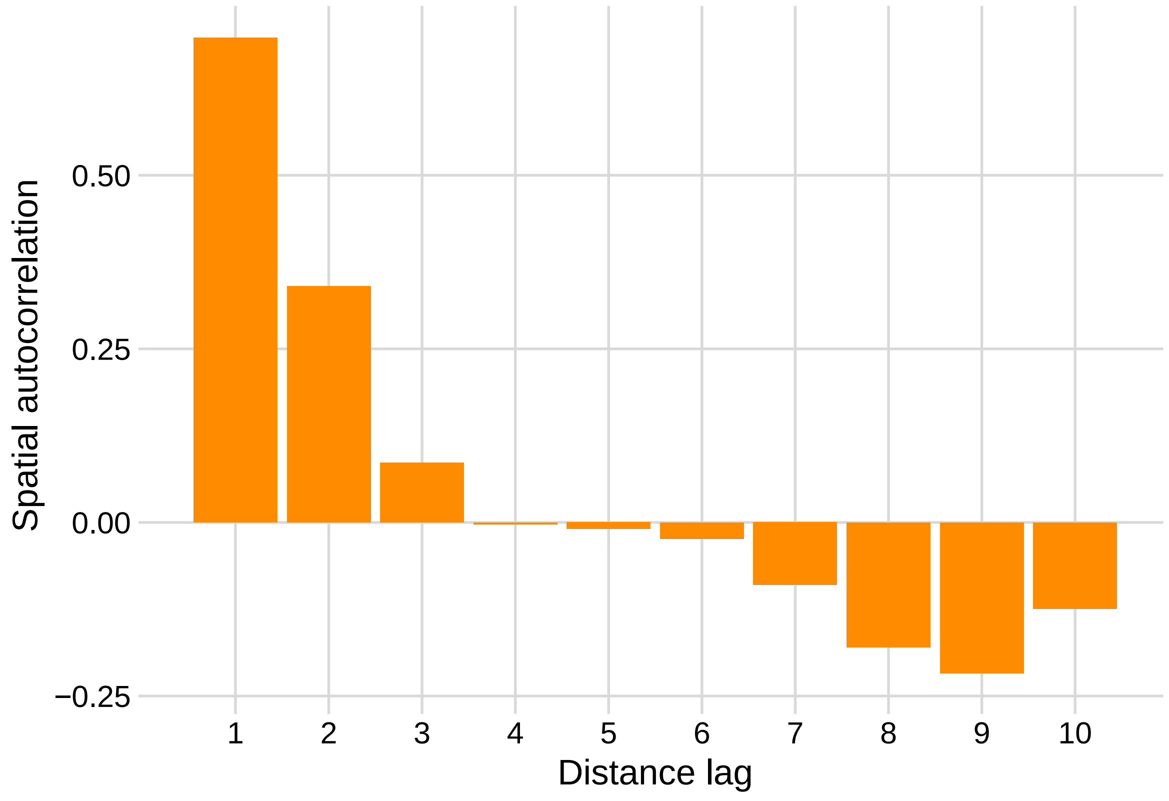
We can generate a correlogram using the output of the `sp.correlogram` function. Note that the figure below is very similar to the one shown in Figure 91.5 in page 269 of the book chapter (Madden et al. 2017c). Let's store the results in a dataframe.

```
df_correl <- data.frame(correl_I$res) %>%
  mutate(lag = c(1:10))

# Show the spatial autocorrelation for 10 distance lags
round(df_correl$X1,3)
```

```
 [1]  0.698  0.340  0.086 -0.002 -0.009 -0.024 -0.090 -0.180 -0.217 -0.124
```

Then, we can generate the plot using ggplot.

```
df_correl %>%
  ggplot(aes(lag, X1))+
  geom_col(fill = "darkorange")+
  theme_minimal_grid()+
  scale_x_continuous(n.breaks = 10)+
  labs(x = "Distance lag", y = "Spatial autocorrelation")
```



3.2.2.2 Semivariance

Semi-variance is a key quantity in geostatistics. This differs from spatial autocorrelation because distances are usually measured in discrete spatial lags. The semi-variance can be defined as half the variance of the differences between all possible points spaced a constant distance apart.

The semi-variance at a distance d = 0 will be zero, because there are no differences between points that are compared to themselves. However, as points are compared to increasingly distant points, the semi-variance increases. At some distance, called the *Range*, the semi-variance will become approximately equal to the variance of the whole surface itself. This is the greatest distance over which the value at a point on the surface is related to the value at another point. In fact, when the distance between two sampling units is small, the sampling units are close together and, usually, variability is low. As the distance increases, so (usually) does the variability.

Results of semi-variance analysis are normally presented as a graphical plot of semi-variance against distance, which is referred to as a semi-variogram. The main characteristics of the semi-variogram of interest are the nugget, the range and the sill, and their estimations are usually based on an appropriate (non-linear) model fitted to the data points representing the semi-variogram.

For the semi-variance, we will use the `variog` function of the *geoR* package. We need the data in the long format (x, y and z). Let's reshape the data to the long format and store it in `epi2` dataframe.

```r
epi2 <-epi %>%
  pivot_longer(2:13,
               names_to = "y",
               values_to = "n") %>%
  mutate(y = as.numeric(y))

head(epi2)
```

```
# A tibble: 6 x 3
      x     y     n
  <dbl> <dbl> <dbl>
1     1     1     2
2     1     2     2
3     1     3     3
4     1     4    33
5     1     5     4
6     1     6     0
```

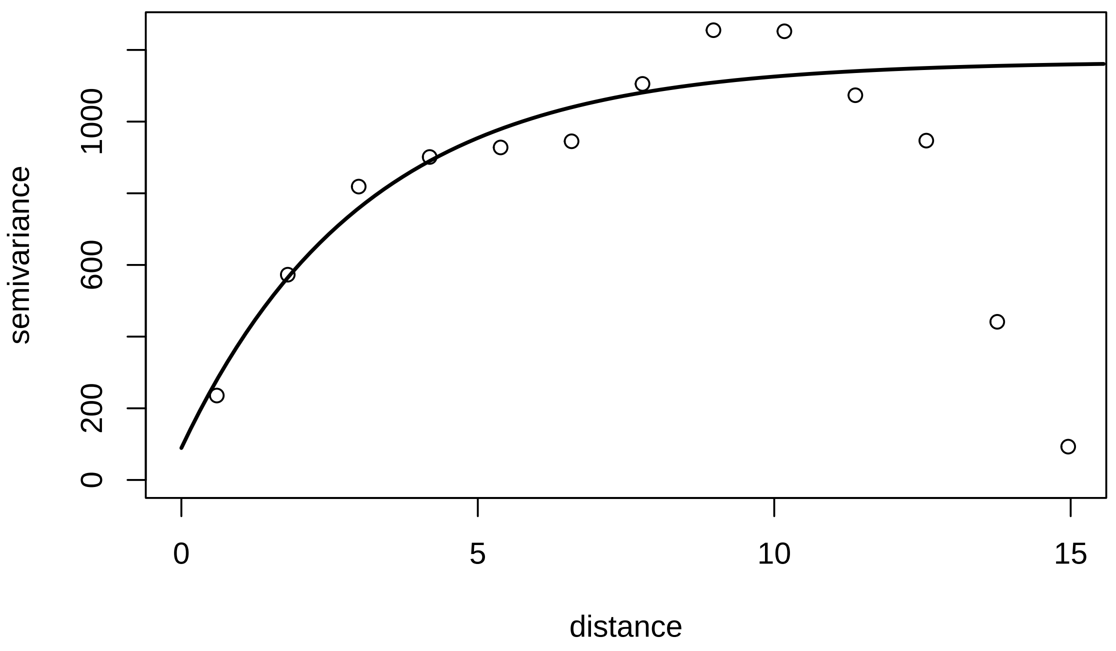
```r
library(geoR)
# the coordinates are x and y and the data is the n
v1 <- variog(coords = epi2[,1:2], data = epi2[,3])
```

```
variog: computing omnidirectional variogram
```

```r
# Model fitting
v2 <- variofit(v1, ini.cov.pars = c(1200, 12),
               cov.model = "exponential",
               fix.nugget = F)
```

```
variofit: covariance model used is exponential
variofit: weights used: npairs
variofit: minimisation function used: optim
```

```r
# Plotting
plot(v1, xlim = c(0,15))
lines(v2, lty = 1, lwd = 2)
```



3.2.2.3 SADIE

SADIE (spatial analysis by distance indices) is an alternative to autocorrelation and semivariance methods described previously, which has found use in plant pathology (Madden et al. 2017c; Xu and Madden 2004; Li et al. 2011). Similar to those methods, the spatial coordinates for the disease intensity (count of diseased individuals) or pathogen propagules values should be provided.

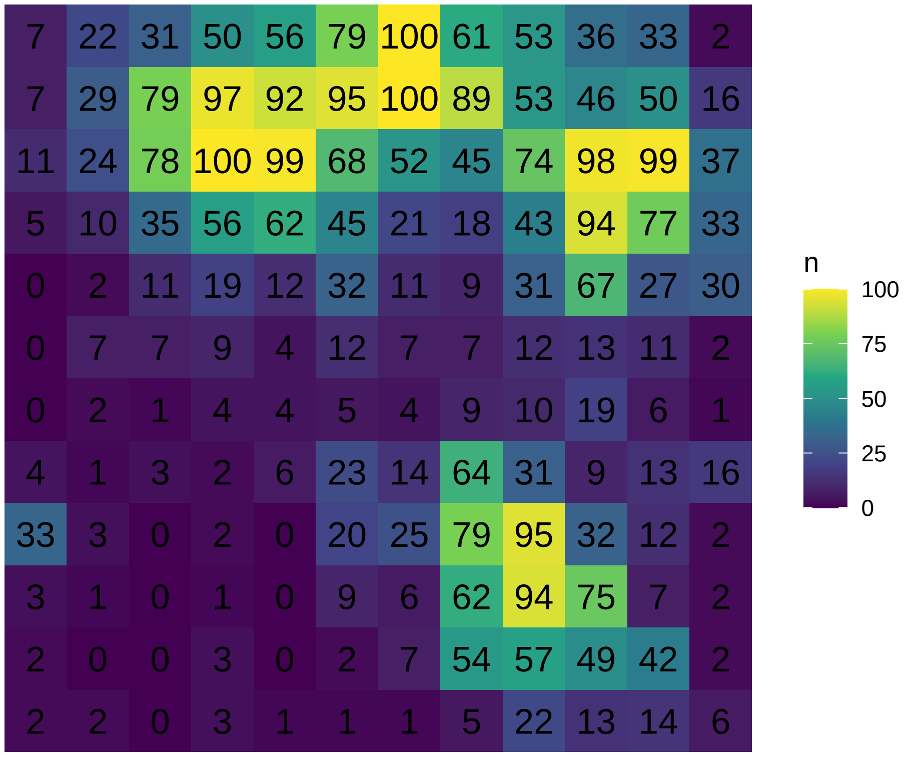
SADIE quantifies spatial pattern by calculating the minimum total distance to regaularity. That is, the distance that individuals must be moved from the starting point defined by the observed counts to the end point at which there is the same number of individuals in each sampling unit. Therefore, if the data are highly aggregated, the distance to regularity will be large, but if the data are close to regular to start with, the distance to regularity will be smaller.

The null hypothesis to test is that the observed pattern is random. SADIE calculates an index of aggregation (Ia). When this is equal to 1, the pattern is random. If this is greater than 1, the pattern is aggregated. Hypothesis testing is based on the randomization procedure. The null hypothesis of randomness, with an alternative hypothesis of aggregation.

An extension was made to quantify the contribution of each sampling unit count to the observed pattern. Regions with large counts are defined as patches and regions with small counts are defined as gaps. For each sampling unit, a clustering index is calculated and can be mapped.

In R, we can use the `sadie` function of the *epiphy* package. The function computes the different indices and probabilities based on the distance to regularity for the observed spatial pattern and a specified number of random permutations of this pattern. To run the analysis, the dataframe should have only three columns: the first two must be the x and y coordinates and the third one the observations. Let's continue working with the simulated epidemic dataset named `epi2`. We can map the original data as follows:

```r
epi2 %>%
  ggplot(aes(x, y, label = n, fill = n))+
  geom_tile()+
  geom_text(size = 5)+
  theme_void()+
  coord_fixed()+
  scale_fill_viridis_c()
```



```r
library(epiphy)
sadie_epi2 <- sadie(epi2)
```

```
Computation of Perry's indices:
```

```r
sadie_epi2
```

```
Spatial Analysis by Distance IndicEs (sadie)

Call:
sadie.data.frame(data = epi2)

Ia: 2.4622 (Pa = < 2.22e-16)
```

The simple output shows the *Ia* value and associated P-value. As suggested by the low P-value, the pattern is highly aggregated. The summary function provides a more complete information such as the overall inflow and outflow measures. A dataframe with the clustering index for each sampling unit is also provided.

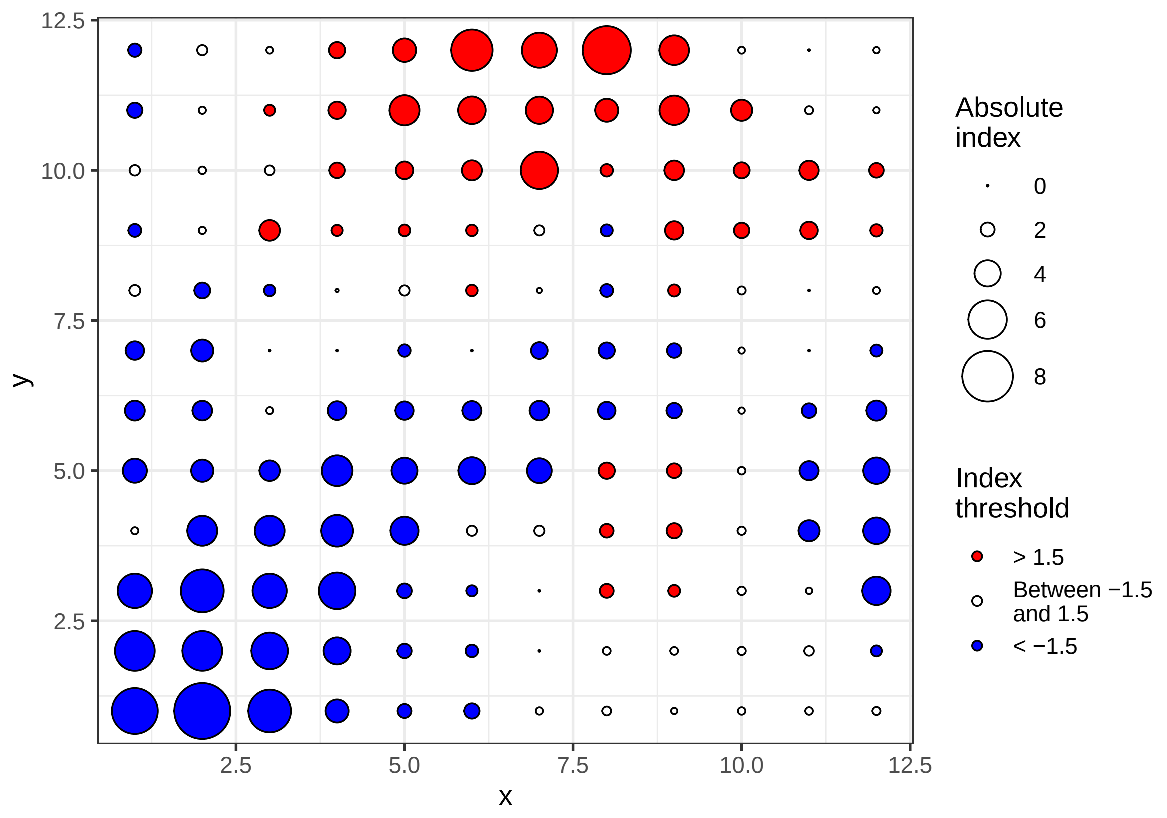
```r
summary(sadie_epi2)
```

```
Call:
sadie.data.frame(data = epi2)

First 6 rows of clustering indices:
  x y  i cost_flows      idx_P idx_LMX prob
1 1 1  2 -11.382725 -7.2242617      NA   NA
2 1 2  2  -9.461212 -6.2258877      NA   NA
3 1 3  3  -7.299482 -5.3390880      NA   NA
4 1 4 33   1.000000  0.8708407      NA   NA
5 1 5  4  -5.830952 -3.6534511      NA   NA
6 1 6  0  -5.301329 -2.9627172      NA   NA

Summary indices:
                      overall    inflow  outflow
Perry's index        2.495346 -2.811023 2.393399
Li-Madden-Xu's index       NA        NA       NA
```

```
Main outputs:
Ia: 2.4622 (Pa = < 2.22e-16)

'Total cost': 201.6062
Number of permutations: 100
```

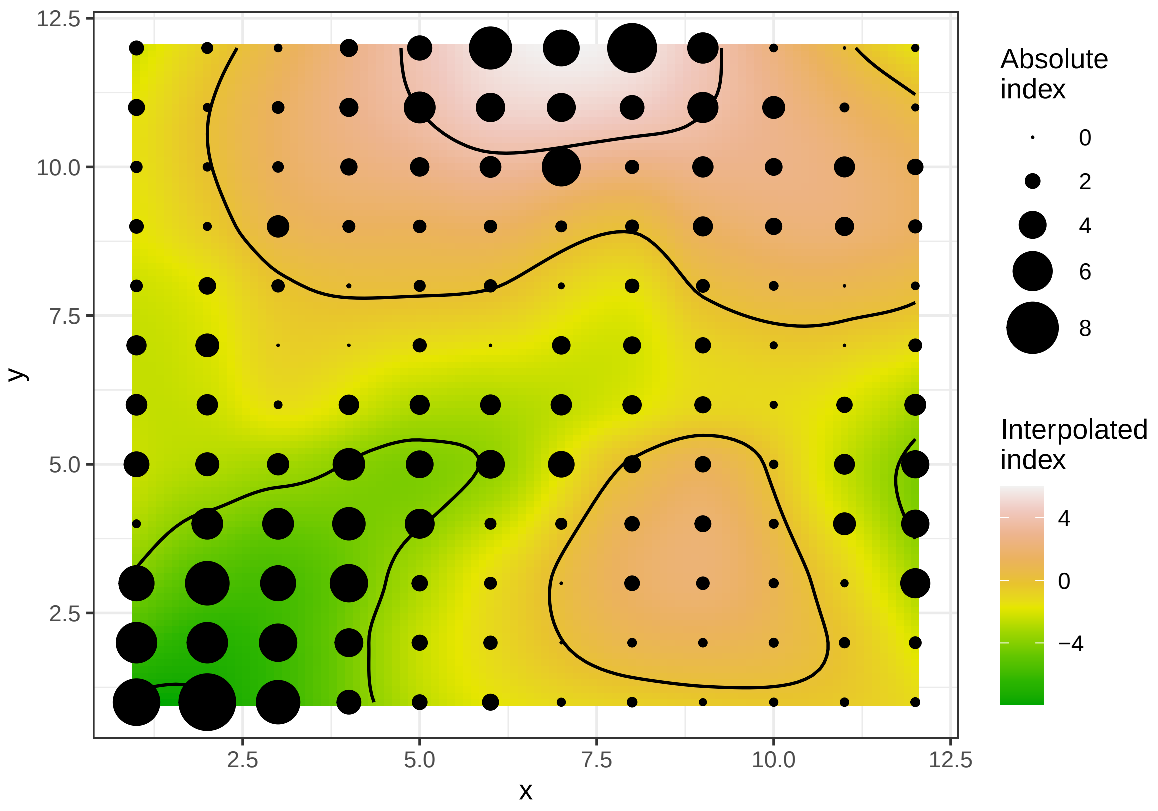
The plot function allows to map the clustering indices and so to identify regions of patches (red, outflow) and gaps (blue, inflow).

```
plot(sadie_epi2)
```



A isocline plot is also possible to obtain by setting the isocline argument as TRUE.

```
plot(sadie_epi2, isoclines = TRUE)
```



12.5
10.0
7.5
5.0
2.5
y
2.5
5.0
7.5
10.0
12.5
x
Absolute index
0
2
4
6
8
Interpolated index
4
0
−4

References

Jeger, M. J., and Viljanen-Rollinson, S. L. H. 2001. The use of the area under the disease-progress curve (AUDPC) to assess quantitative disease resistance in crop cultivars. Theoretical and Applied Genetics. 102:32–40 Available at: http://dx.doi.org/10.1007/s001220051615.

Li, B., Madden, L. V., and Xu, X. 2011. Spatial analysis by distance indices: an alternative local clustering index for studying spatial patterns. Methods in Ecology and Evolution. 3:368–377 Available at: http://dx.doi.org/10.1111/j.2041-210x.2011.00165.x.

Madden, L. V., Hughes, G., and Bosch, F. van den. 2017a. The study of plant disease epidemics. Available at: http://dx.doi.org/10.1094/9780890545058.

Madden, L. V., Hughes, G., and van den Bosch, F., eds. 2017b. CHAPTER 4: Temporal analysis i: Quantifying and comparing epidemics. In The American Phytopathological Society, p. 63–116. Available at: http://dx.doi.org/10.1094/9780890545058.004.

Madden, L. V., Hughes, G., and van den Bosch, F., eds. 2017c. CHAPTER 9: Spatial aspects of epidemics—III: Patterns of plant disease. In The American Phytopathological Society, p. 235–278. Available at: http://dx.doi.org/10.1094/9780890545058.009.

Simko, I., and Piepho, H.-P. 2012. The Area Under the Disease Progress Stairs: Calculation, Advantage, and Application. Phytopathology®. 102:381–389 Available at: http://dx.doi.org/10.1094/phyto-07-11-0216.

Xu, X.-M., and Madden, L. V. 2004. Use of SADIE statistics to study spatial dynamics of plant disease epidemics. Plant Pathology. 53:38–49 Available at: http://dx.doi.org/10.1111/j.1365-3059.2004.00949.x.