R4PDE

Emerson M. Del Ponte

Table of contents

Welcome

R for Plant Disease Epidemiology (R4PDE) is a book project in the early stage of development. 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. 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 to run the examples.

This book is not a resource to learn data science using R as there are excellent books such as the R for data science. If you read Portuguese (as many of the students of my course), I strongly recommend Análises Ecológicas no R and Software R para avaliação de dados experimentais as excellent resources for learning R (and also statistics with R).

Throughout the book, I use several general and a few specific R packages for conducting the most common analysis of plant disease epidemiology data, in particular epifitter and epiphy. In most part, I use data and reproduce some of the analyses shown in the book *The study of Plant Disease Epidemics* (Madden et al. 2017c).

This is a work in progress, meaning that it is under frequent technical and copy editing. This website is free to use, and is licensed under a Creative Commons licence. The codes for reproducing all analyses shown here are available on GitHub. There are no immediate plans to publish a physical copy of the book, but I may consider it in the future as the book takes shape. This website is hosted by https://www.netlify.com/.

Please note that R4PDE uses a Contributor Code of Conduct. By contributing to this book, you agree to abide by its terms.

Acknowledgements

To those who have contributed fixes and improvements via pull request: Adam Sparks (@adamhsparks)

About the author

Emerson M. Del Ponte is an Associate Professor at the Departamento de Fitopatologia, Universidade Federal de Viçosa in Brazil. He holds a DSc in Plant Pathology (Universidade Federal de Pelotas, 2004) with a one-year visit to Cornell University (Bergstrom Lab). He worked during almost two years as a postdoctoral associate in a project involving disease risk assessment and prediction (Yang Lab, Iowa State University) prior to join the Universidade Federal do Rio Grande do Sul, Brazil, as assistant professor of plant pathology. His peer-reviewed publications can be found in the following academic websites: ORCID, Google Scholar, and ResearchGate.

Emerson is a strong advocate for an open and reproducible research model and culture that may ultimately contribute to a more accessible, transparent and reliable science. This led him to co-found Open Plant Pathology initiative together with Adam Sparks. In his Lab, students make use of R language for all statistical and data science related activities. All data and computational codes produced during the research are made available ahead of peer-review. The code can be found on GitHub. The research outcomes are organized and shared as a research compendium and preprints of manuscripts are archived in Open Science Framework (OSF).

1 Introduction

> This is a work in progress that is currently undergoing heavy technical editing and copy-editing

A diseased plant results from the combination of three elements: susceptible host plant, a virulent pathogen, and favorable environmental conditions. However, when a pathogen population establishes and causes disease in a host population, the phenomenon is called an *epidemic*, or the disease in populations. Among several definitions of epidemic, a comprehensive one is the *change in disease intensity in a host population over time and space* (Madden et al. 2017c).

Epidemics are economically important because of their potential to reduce crop yields, lower product quality, and increase control costs, depending on the level of intensity. In fact, several examples of famous epidemics, that reached pandemic levels and caused devastating effects to crops, can be found in the literature (Agrios 2005).

> Note

A famous example of pandemics is the Irish potato famine of 1845–1847 caused by the late blight pathogen (*Phytophthora infestans*), the cause of a disease that changed the history of Europe and the United States, and that is also known for its role in developing the science of plant pathology. During the 1840s, the disease devastated potato crops, a staple food of the Irish at the time. The disease was triggered by the introduction of a new virulent pathogen population that found the right environmental conditions (cool and wet weather) for infection and development in a highly dense population of susceptible hosts.

Botanical epidemiology, or the study of plant disease epidemics, is a relatively young discipline formally recognized in 1963. Two important facts marked that year. J.E. Vanderplank published his famous book "Plant Disease Epidemic and Control" (Vanderplank 1963) and the first International Epidemiology Workshop was held in Pau, France. Vanderplank is fully acknowledged as the founding father of plant disease epidemiology (Zadoks and Schein 1988; Thresh 1998)

Part I

Epidemic data

2 Disease variables

> This is a work in progress that is currently undergoing heavy technical editing and copy-editing

2.1 Phytopathometry

Studies on the progress of epidemics in time or their spread in space cannot be conducted without data collected in the field - or in same cases simulated. The study of plant disease quantification is known as Phytopathometry, a branch of plant pathology tasked with the science of disease measurement, but which has strong roots in epidemiology (Bock et al. 2021). Historically, disease quantification has been performed visually, but advances in both imaging and remote sensing technologies have directly impacted the field during the last several decades. Therefore, the quantity of disease can be obtained via estimation (visually by human eye) or measurement (sensor or digital technologies: RGB, MSI, HSI). This means that several variables can be used to express disease occurrence and quantity.

While measuring disease is a a more objective task, visual assessment is largely subjective and, as such, known to vary among human raters. This occurs because raters vary in their inherent abilities, training, or are more or less affected by the chosen method (e.g. scales). Disease is estimated or measured on a specimen in a population, or on a sample of specimens drawn from a population. The specimen can be a plant organ, an individual plant, a group of plants, a field or a farm, and these also dictate how terms are defined to refer to disease quantity.

Finally, when developing new or improving existing disease assessment methods, it is important to assess how close the estimations or measurements are from reference (gold standard) values. There are several methods that can be used to assess reliability, precision and accuracy of estimates or measures. The choice depends on the objective of the work but largely on the type or nature of the data, as it will be discussed further.

2.2 Terminology

A general term used to refer to the quantity of disease expressed by any means is **disease intensity**. This term has little or no practical value as it only suggest that the disease is more

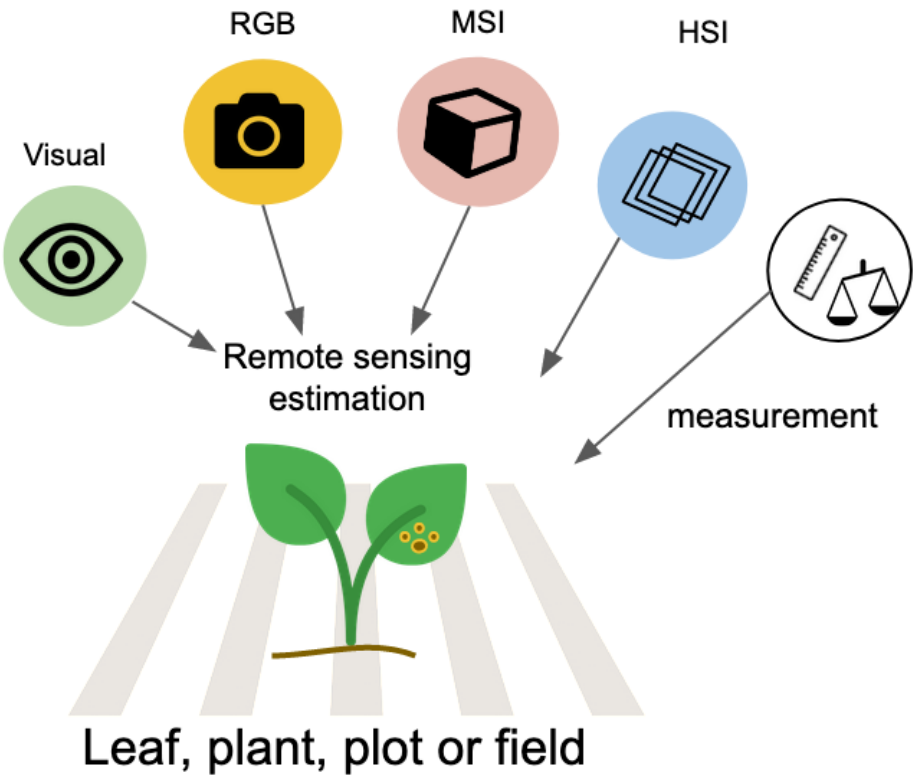


Figure 2.1: Different approaches used to obtain estimates or measures of plant disease

or less "intense". We need more specific terms to refer to disease quantity. A primary task in disease assessment is to classify each specimen, usually in a sample or in the population, as diseased or not diseased. This binary (yes/no) assessment may be sufficient to express disease intensity if the goal is to assess, for example, the number or proportion of diseased specimens in a sample or a population of specimens.

The above leads to two terms: disease **incidence** and **prevalence**. While incidence is commonly used to refer to the proportion or number (count) of plants (or their organs) as the observational units at the field scale or below, prevalence is used when referring to the proportion or number of fields or farms with diseased plants in a large production area or region (Nutter et al. 2006). Hence, prevalence is equivalent to incidence, only differing in the spatial scale of the sampling unit.

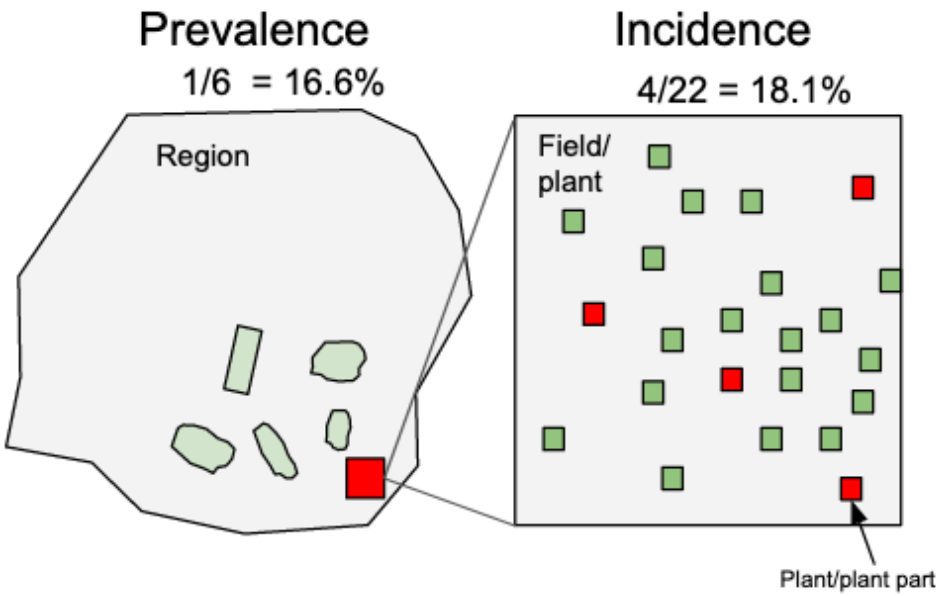


Figure 2.2: Schematic representation of prevalence and incidence of plant diseases

In many cases we need to ascertain the *degree to which a specimen is diseased*, which is a definition of disease **severity**. Elsewhere, severity is defined restrictively as the proportion of the unit that is symptomatic (Nutter et al. 2006). However, a broader view of severity encompasses other metrics including lesion counts and scores based on ordinal scales. These latter scales can be further divided into classes defined based on either the percentage scale or descriptions of symptoms (Bock et al. 2021). In some cases disease is expressed in terms of (average) lesion size or area, which can be considered a measure of severity. It is commonly used to determine host resistance, pathogen aggressiveness or environmental influence.

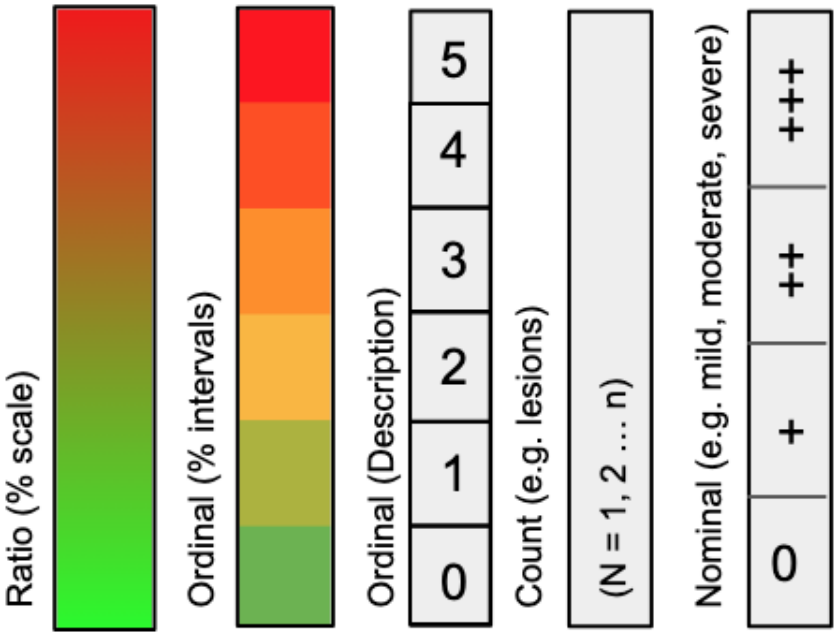


Figure 2.3: Different scales and variable types used to represent severity of plant diseases

3 Data types and distributions

> This is a work in progress that is currently undergoing heavy technical editing and copy-editing

The data used to express disease as incidence or any kind of severity measures vary in their nature as they can be discrete or continuous.

Discrete variables are countable (involve integers) at a finite amount of time. That is, only a limited number of values (nominal or ordinal) is possible and these cannot be subdivided into parts. For example, a plant or plant part can be either disease or not diseased (nominal data). One can't count 1.5 diseased plants. Also, a plant classified as diseased may exhibit a certain number of lesions (count data) or be classified into a specific class of severity (ordinal data, common in ordinal scales, e.g., 1-9). Disease data in the form of counts usually relate to the number of infections per sampling units. Most commonly, counts refer to the pathogen population that is assessed, such as number of airborne or soilborne propagules.

A **continuous** variable, different from discrete, can be measured on a scale and can have any numeric value between two numbers. For example, the size of a lesion on a plant can be measured at a very precise scale (cm or mm). An estimate of severity in percent scale (% diseased area) can take any value between non zero and 100%. Although discrete at the individual level, incidence at the sample level can be treated as continuous, as it can take any value in proportion or percentage.

The disease variables can also be described by a statistical distribution, which are models that give the probability that a particular value (or a range of values) will be drawn from a specific distribution. Knowledge about statistical or mathematical distributions constitute an important step to improve our understanding of data-collection methods, designs of experiments and data analysis such as data summarization or hypothesis testing.

3.1 Statistical distributions

3.1.1 Binomial distribution

For incidence (and prevalence), the data is binary at the individual level, as there are only two possible outcomes in a *trial*: the plant or plant part is disease or not diseased. The

statistical distribution that best describe the incidence data at the individual level is the *binomial distribution*.

Let's simulate the binomial outcomes for a range of probabilities in a sample of 100 units, using the `rbinom()` function in R. For a single trial (e.g., status of plants in a single plant row), the `size` argument is set to 1.

```
library(tidyverse)
theme_set(theme_bw(base_size = 14)) # set global theme

set.seed(123) # for reproducibility
P.1 <- rbinom(100, size = 1, prob = 0.1)
P.3 <- rbinom(100, size = 1, prob = 0.3)
P.7 <- rbinom(100, size = 1, prob = 0.7)
P.9 <- rbinom(100, size = 1, prob = 0.9)
binomial_data <- data.frame(P.1, P.3, P.7, P.9)
```

We can then visualize the plots.

```
binomial_data |>
  pivot_longer(1:4, names_to = "P",
               values_to = "value") |>
  ggplot(aes(value)) +
  geom_histogram(fill = "darkorange",
                 bins = 10) +
  facet_wrap( ~ P)
```

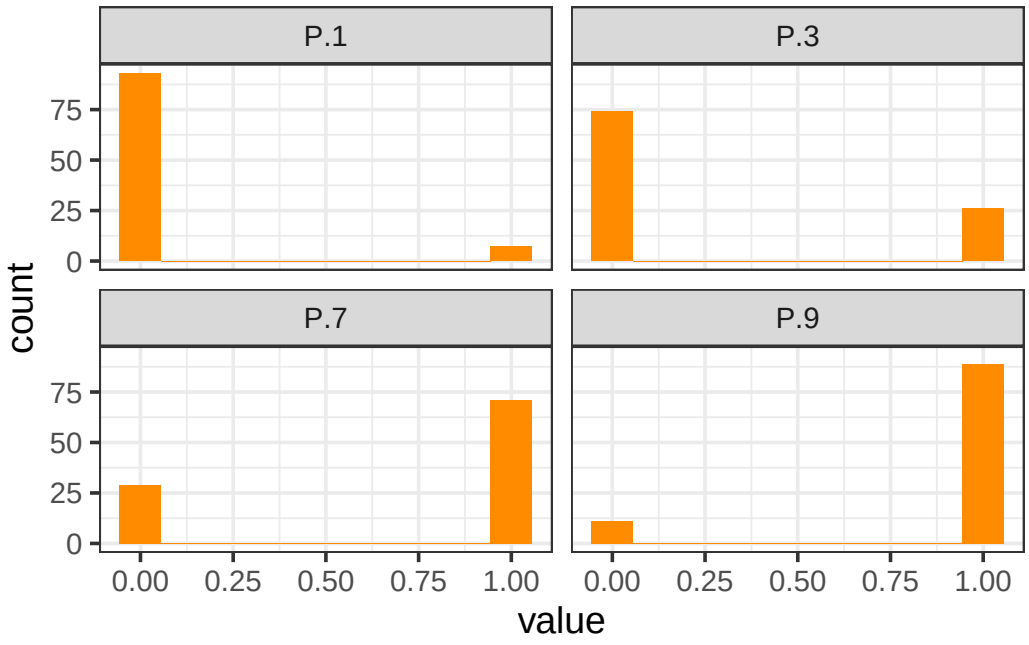


Figure 3.1: Binomial distribution to describe binary data

3.1.2 Beta distribution

Disease incidence (or prevalence) at the sample or population level can be expressed as proportion of diseased individuals. The same applies to disease severity when expressed as proportion of the organ area affected (a ratio variable). For such cases, the beta distribution, which is bounded between 0 and 1, provides a good description. Let's simulate some data using the `rbeta()` function.

```
beta1.5 <- rbeta(n = 1000, shape1 = 1, shape2 = 5)
beta5.5 <- rbeta(n = 1000, shape1 = 5, shape2 = 5)
beta_data <- data.frame(beta1.5, beta5.5)
```

Notice that there are two shape parameters in the beta distribution: `shape1` and `shape2` to be defined. This makes the distribution very flexible and with different potential shapes as we can see below.

```
library(tidyverse)
theme_set(theme_bw(base_size = 14)) # set global theme

beta_data |>
```

```r
pivot_longer(1:2, names_to = "P",
             values_to = "value") |>
ggplot(aes(value)) +
geom_histogram(fill = "darkorange",
               color = "white",
               bins = 15) +
scale_x_continuous(limits = c(0, 1)) +
facet_wrap( ~ P)
```

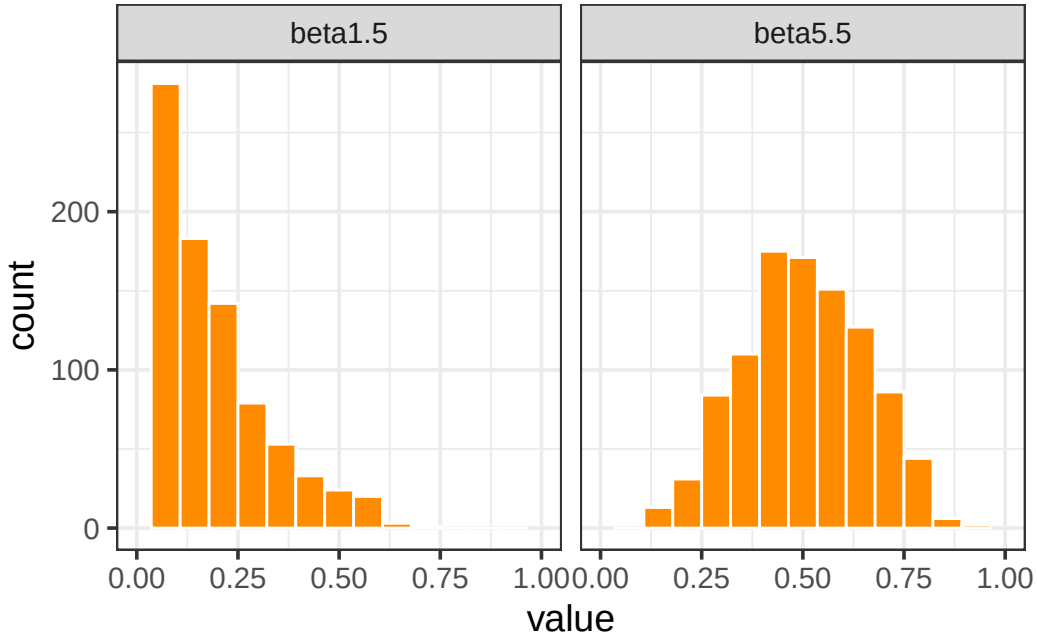


Figure 3.2: Binomial distribution to describe proportion data

3.1.3 Poisson distribution

The number of diseased plants, plant parts or individual symptoms (lesions) are discrete variables (integers) which cannot take negative values. These can be described by a Poisson distribution, a discrete distribution that counts the number of events in a Poisson process. In R, we can used the `rpois()` function to obtain 100 random observations following a Poisson distribution. For such, we need to inform the number of observation (n = 100) and `lambda`, the vector of means.

```r
poisson5 <- rpois(100, lambda = 10)
poisson35 <- rpois(100, lambda = 35)
poisson_data <- data.frame(poisson5, poisson35)
```

```r
poisson_data |>
  pivot_longer(1:2, names_to = "P",
               values_to = "value") |>
  ggplot(aes(value)) +
  geom_histogram(fill = "darkorange",
                 color = "white",
                 bins = 15) +
  facet_wrap( ~ P)
```

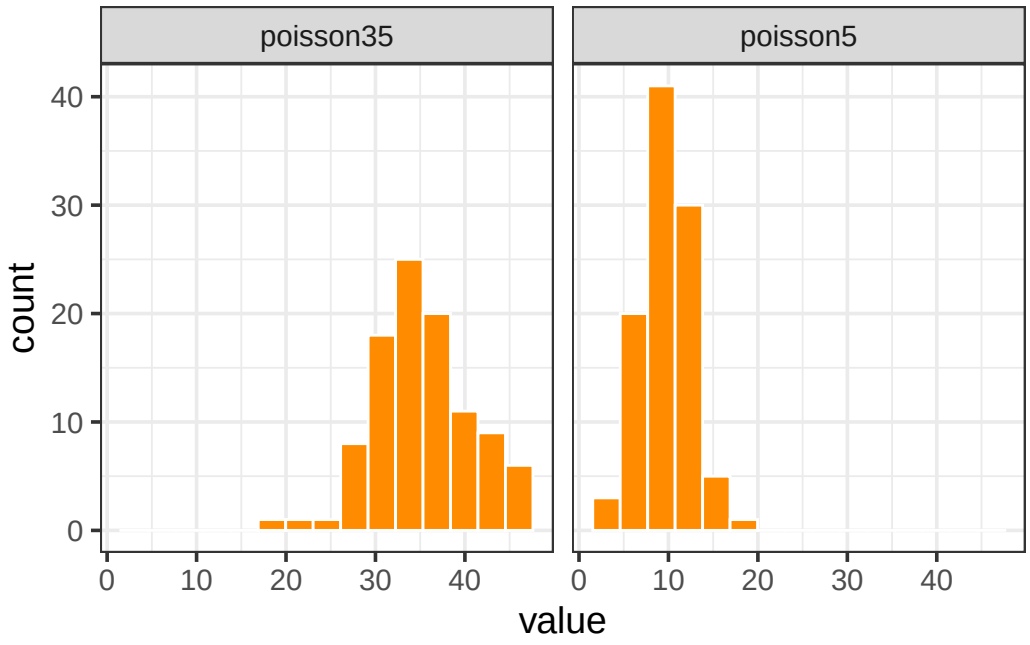


Figure 3.3: Binomial distribution to describe count data

3.1.4 Gamma distribution

When working with a continuous variables, such as lesion size, these random variables are usually described by the normal distribution. However, the problem is that the normal (Gaussian) distribution includes negative values, an unrealistic situation. Therefore, we can use the gamma distribution, which cannot take negative values, to simulate continuous plant disease

data. We can use the `rgamma()` function that requires the number of samples (n = 100 in our case) and the `shape`, or the mean value.

```
gamma10 <- rgamma(n = 100, shape = 10, scale = 1)
gamma35 <- rgamma(n = 100, shape = 35, scale = 1)
gamma_data <- data.frame(gamma10, gamma35)
```

```
gamma_data |>
  pivot_longer(1:2, names_to = "P",
               values_to = "value") |>
  ggplot(aes(value)) +
  geom_histogram(fill = "darkorange",
                 color = "white",
                 bins = 15) +
  ylim(0, max(gamma_data$gamma35)) +
  facet_wrap( ~ P)
```

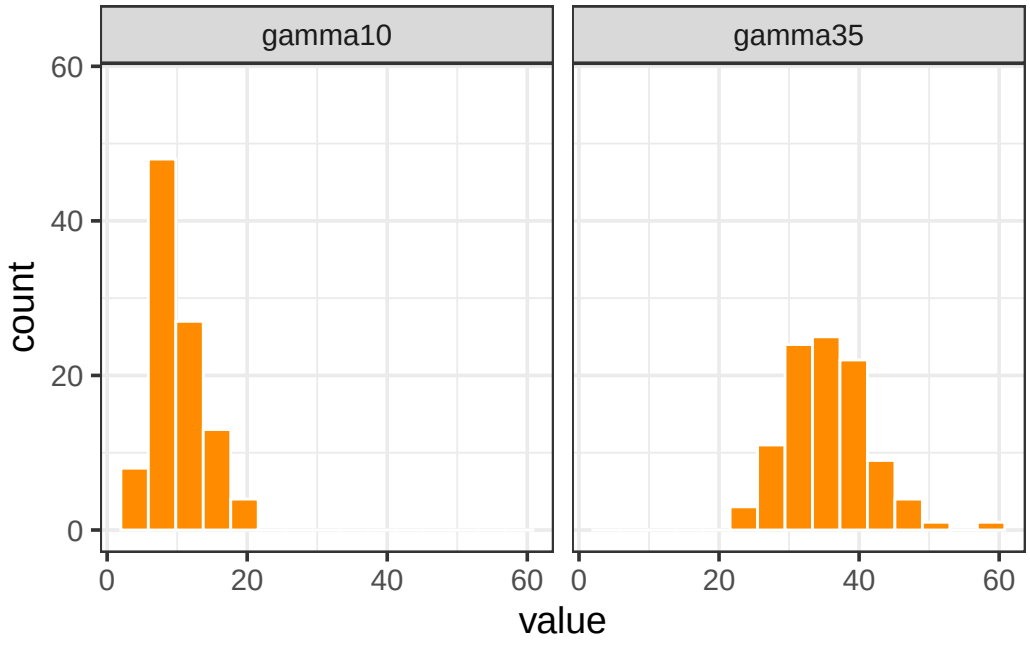


Figure 3.4: Gamma distribution to describe continuous data

4 Reliability and accuracy

> This is a work in progress that is currently undergoing heavy technical editing and copy-editing

Disease severity, mainly when expressed in percent area diseased assessed visually, is acknowledged as a more difficult and less time- and cost-effective plant disease variable to obtain. However, errors may occur even when assessing a more objective measure such as incidence. This is the case when an incorrect assignment or confusion of symptoms occur. In either case, the quality of the assessment of any disease variable is very important and should be gauged in the studies. Several terms can be used when evaluating the quality of disease assessments, including reliability, precision, accuracy or agreement.

Reliability: The extent to which the same estimates or measurements of diseased specimens obtained under different conditions yield similar results. There are two types. The *inter-rater reliability* (or reproducibility) is a measure of consistency of disease assessment across the same specimens between raters or devices. The *intra-rater* reliability (or repeatability) measures consistency by the same rater or instrument on the same specimens (e.g. two assessments in time by the same rater).

Precision: A statistical term to express the measure of variability of the estimates or measurements of disease on the same specimens obtained by different raters (or instruments). However, reliable or precise estimates (or measurements) are not necessarily close to an actual value, but precision is a component of accuracy or agreement.

Accuracy or agreement: These two terms can be treated as synonymous in plant pathological research. They refer to the closeness (or concordance) of an estimate or measurement to the actual severity value for a specimen on the same scale. Actual values may be obtained using various methods, against which estimates or measurements using an experimental assessment method are compared.

An analogy commonly used to explain accuracy and precision is the archer shooting arrows at a target and trying to hit the bull's eye (center of the target) with each of five arrows. The figure below is used to demonstrate four situations from the combination of two levels (high and low) for precision and accuracy. The figure was produced using the `ggplot` function of *ggplot2* package.

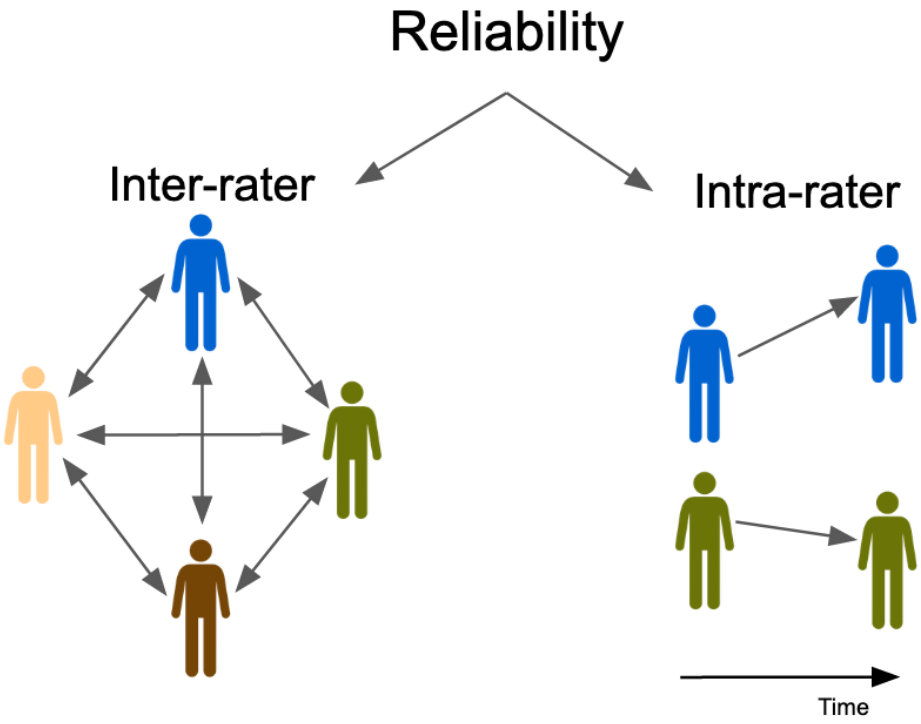


Figure 4.1: Two types of reliability of estimates or measures of plant disease intensity

```r
library(ggplot2)
target <-
  ggplot(data.frame(c(1:10),c(1:10)))+
  geom_point(aes(x = 5, y = 5), size = 71.5, color = "black")+
  geom_point(aes(x = 5, y = 5), size = 70, color = "orange")+
  geom_point(aes(x = 5, y = 5), size = 60, color = "white")+
  geom_point(aes(x = 5, y = 5), size = 50, color = "orange")+
  geom_point(aes(x = 5, y = 5), size = 40, color = "white")+
  geom_point(aes(x = 5, y = 5), size = 30, color = "orange")+
  geom_point(aes(x = 5, y = 5), size = 20, color = "white")+
  geom_point(aes(x = 5, y = 5), size = 10, color = "orange")+
  geom_point(aes(x = 5, y = 5), size = 4, color = "white")+
  ylim(0,10)+
  xlim(0,10)+
  theme_void()

hahp <- target +
  labs(subtitle = "High Accuracy High Precision")+
  theme(plot.subtitle = element_text(hjust = 0.5))+
  geom_point(aes(x = 5, y = 5), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 5, y = 5.2), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 5, y = 4.8), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 4.8, y = 5), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 5.2, y = 5), shape = 4, size =2, color = "blue")


lahp <- target +
  labs(subtitle = "Low Accuracy High Precision")+
  theme(plot.subtitle = element_text(hjust = 0.5))+
  geom_point(aes(x = 6, y = 6), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 6, y = 6.2), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 6, y = 5.8), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 5.8, y = 6), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 6.2, y = 6), shape = 4, size =2, color = "blue")


halp <- target +
  labs(subtitle = "High Accuracy Low Precision")+
  theme(plot.subtitle = element_text(hjust = 0.5))+
  geom_point(aes(x = 5, y = 5), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 5, y = 5.8), shape = 4, size =2, color = "blue")+
```

```
  geom_point(aes(x = 5.8, y = 4.4), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 4.4, y = 5), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 5.6, y = 5.6), shape = 4, size =2, color = "blue")

lalp <- target +
  labs(subtitle = "Low Accuracy Low Precision")+
  theme(plot.subtitle = element_text(hjust = 0.5))+
  geom_point(aes(x = 5.5, y = 5.5), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 4.5, y = 5.4), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 5.2, y = 6.8), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 4.8, y = 3.8), shape = 4, size =2, color = "blue")+
  geom_point(aes(x = 5.2, y = 3), shape = 4, size =2, color = "blue")


library(patchwork)
(hahp | lahp) /
(halp | lalp)
```

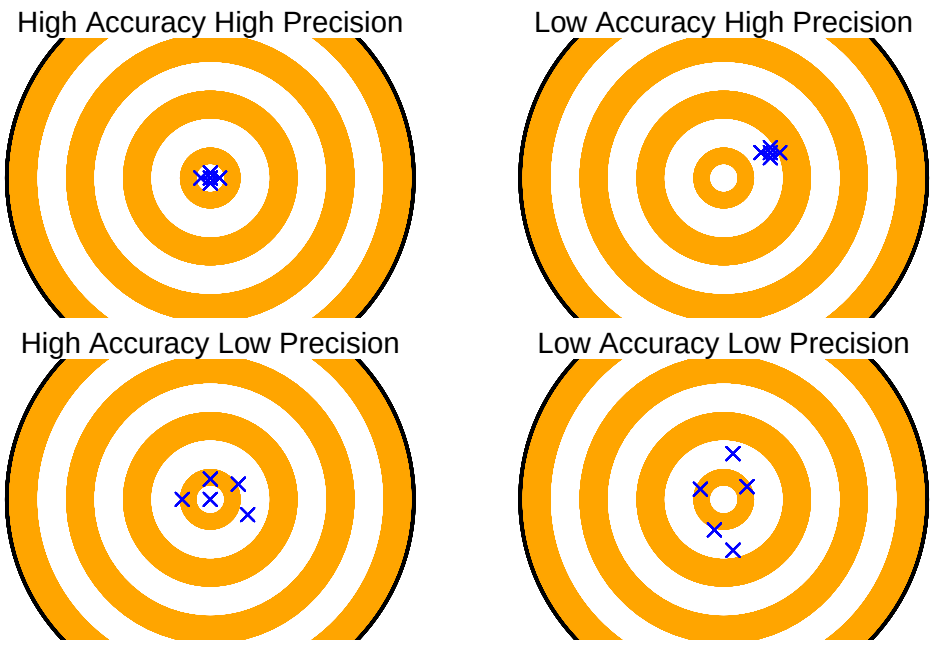


Figure 4.2: The accuracy and precision of the archer is determined by the location of the group of arrows

Another way to visualize accuracy and precision is via scatter plots for the relationship between

the actual values and the estimates.

```
library(tidyverse)
theme_set(theme_bw())
dat <-
tibble::tribble(
  ~actual,   ~ap,   ~ip,   ~ai,   ~ii,
        0,     0,    10,     0,    25,
       10,    10,    20,     5,    10,
       20,    20,    30,    30,    10,
       30,    30,    40,    30,    45,
       40,    40,    50,    30,    35,
       50,    50,    60,    60,    65,
       60,    60,    70,    50,    30
  )

ap <- dat |>
  ggplot(aes(actual, ap))+
  geom_abline(intercept = 0, slope = 1,
              linetype = 2, size = 1)+
    geom_smooth(method = "lm")+
   geom_point(color = "orange", size = 3)+
   ylim(0,70)+
  xlim(0,70)+
  labs(x = "Actual", y = "Estimate",
       title = "High Acccuracy High Precision")

ip <- dat |>
  ggplot(aes(actual, ip))+
  geom_abline(intercept = 0, slope = 1,
              linetype = 2, size = 1)+
  geom_smooth(method = "lm", se = F)+
  geom_point(color = "orange", size = 3)+
  ylim(0,70)+
  xlim(0,70)+
  labs(x = "Actual", y = "Estimate",
       title = "Low Acccuracy High Precision")

ai <- dat |>
  ggplot(aes(actual, ai))+
  geom_abline(intercept = 0, slope = 1,
              linetype = 2, size = 1)+
```

```
  geom_smooth(method = "lm", se = F)+
  geom_point(color = "orange", size = 3)+
  ylim(0,70)+
  xlim(0,70)+
  labs(x = "Actual", y = "Estimate",
       title = "High Acccuracy Low precision")

ii <- dat |>
  ggplot(aes(actual, ii))+
  geom_abline(intercept = 0, slope = 1,
              linetype = 2, size = 1)+
  geom_smooth(method = "lm", se = F)+
  geom_point(color = "orange", size = 3)+
  ylim(0,70)+
  xlim(0,70)+
  labs(x = "Actual", y = "Estimate",
       title = "Low Acccuracy Low Precision")

library(patchwork)
(ap | ip) / (ai | ii)
```

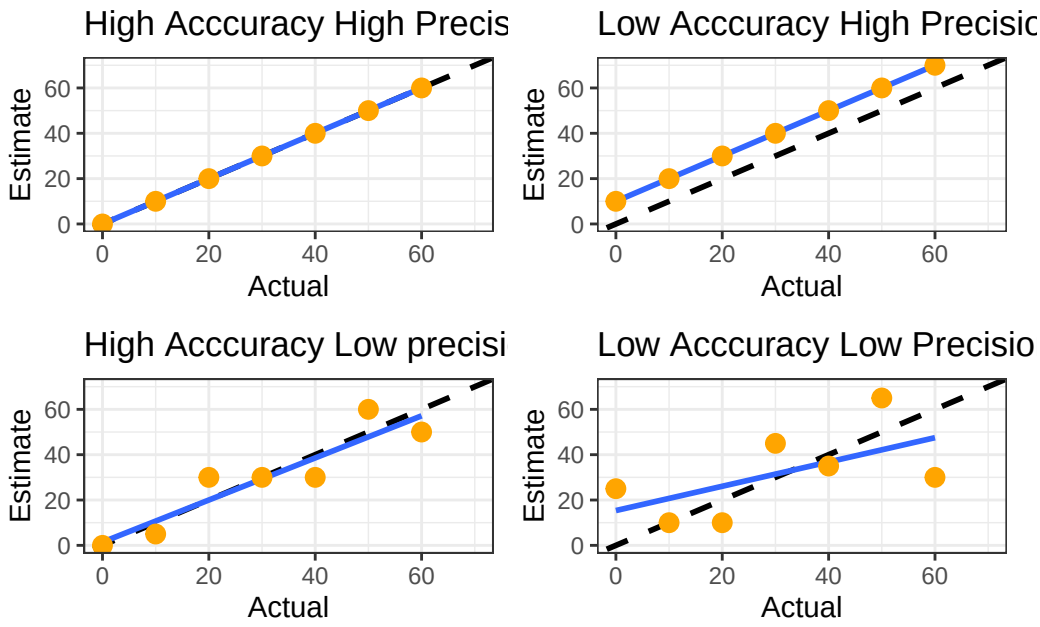


Figure 4.3: Scatter plots for the relationship between actual and estimated values representing situations of low or high precision and accuracy. The dashed line indicates the perfect concordance and the solid blue line represents the fit of the linear regression model

4.1 Statistical summaries

A formal assessment of the quality of estimates or measures is made using statistical summaries of the data expressed as indices that represent reliability, precision and accuracy. These indices can further be used to test hypothesis such as if one or another method is superior than the other. The indices or the tests vary according to the nature of the variable, whether continuous, binary or categorical.

4.1.1 Inter-rater reliability

To calculate measures of inter-rater reliability (or reproducibility) we will work with a fraction of a larger dataset used in a published study. There, the authors tested the effect of standard area diagrams (SADs) on the reliability and accuracy of visual estimates of severity of soybean rust.

The selected dataset consists of five columns with 20 rows. The first is the leaf number and the others correspond to assessments of percent soybean rust severity by four raters (R1 to

R4). Each row correspond to one symptomatic leaf. Let's assign the tibble to a dataframe called `sbr` (an acronym for soybean rust). Note that the variable is continuous.

```
library(tidyverse)
sbr <- tribble(
~leaf, ~R1, ~R2,  ~R3, ~R4,
1, 0.6, 0.6,  0.7, 0.6,
2,   2, 0.7,    5,   1,
3,   5,   5,    8,   5,
4,   2,   4,    6,   2,
5,   6,  14,   10,   7,
6,   5,   6,   10,   5,
7,  10,  18, 12.5,  12,
8,  15,  30,   22,  10,
9,   7,   2,   12,   8,
10,  6,   9, 11.5,   8,
11,  7,   7,   20,   9,
12,  6,  23,   22,  14,
13, 10,  35, 18.5,  20,
14, 19,  10,    9,  10,
15, 15,  20,   19,  20,
16, 17,  30,   18,  13,
17, 19,  53,   33,  38,
18, 17, 6.8,   15,   9,
19, 15,  20,   18,  16,
20, 18,  22,   24,  15
        )
```

Let's explore the data using various approaches. First, we can visualize how the individual estimates by the raters differ for a same leaf.

```
# set the global theme
theme_set(theme_bw())
library(ggthemes)

# transform from wide to long format
sbr2 <- sbr |>
  pivot_longer(2:5, names_to = "rater",
               values_to = "estimate")

# create the plot
sbr2 |>
```

```
  ggplot(aes(leaf, estimate, color = rater,
             group = leaf))+
  geom_line(color = "black")+
  geom_point(size = 2)+
  scale_color_colorblind()+
  labs(y = "Severity estimate (%)",
       x = "Leaf number",
       color = "Rater")
```

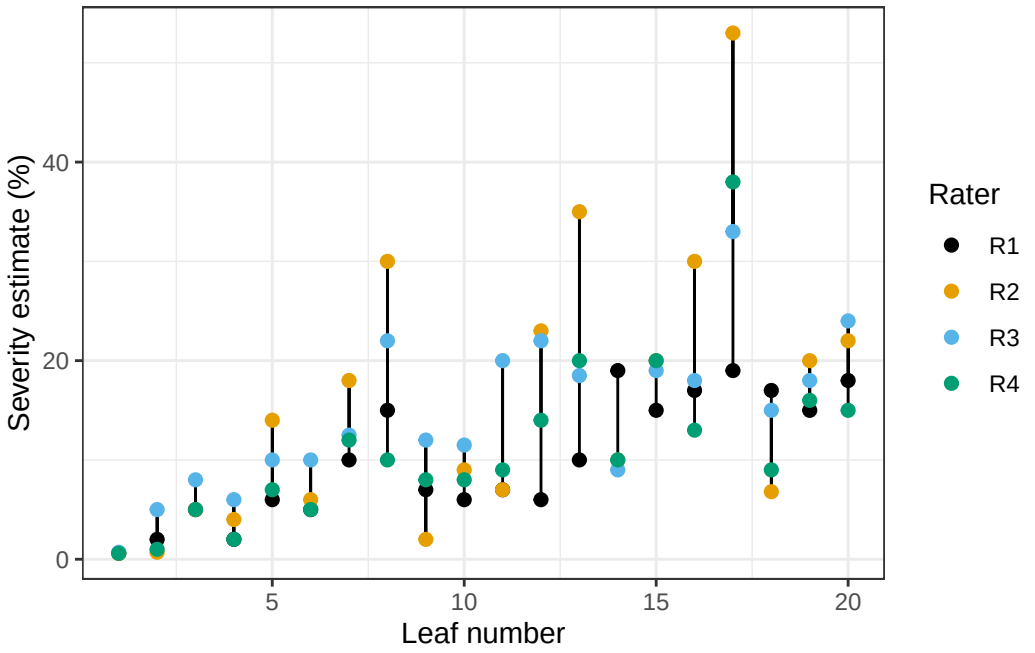


Figure 4.4: Visual estimates of soybean rust severity for each leaf by each of four raters

Alternatively, we can visualize the distribution of the estimates by rater using boxplots.

```
sbr2 |>
  ggplot(aes(rater, estimate))+
  geom_boxplot(outlier.colour = NA, width =0.5)+
  geom_jitter(width = 0.1,shape = 1, size =2)+
  labs(y = "Severity estimate (%)",
       x = "Rater")
```

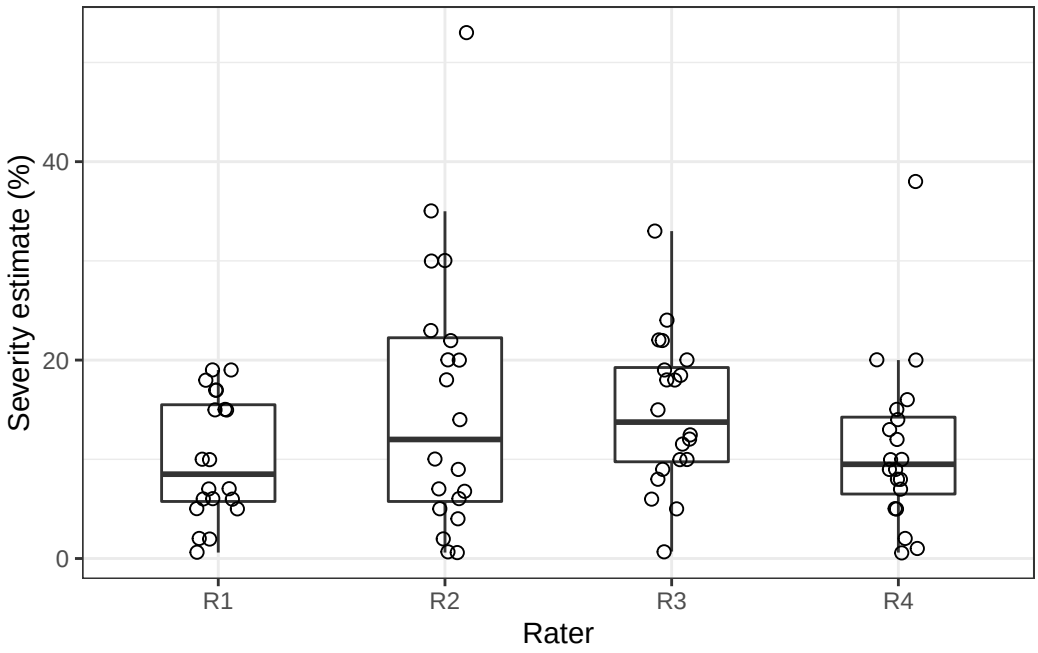


Figure 4.5: Distribution of severity estimates of soybean rust by rater

Another interesting visualization is the correlation matrix of the estimates between all possible pair of raters. The `ggpairs` function of the *GGally* package is handy for this task.

```
library(GGally)
theme_set(theme_light())

# create a new dataframe with only raters
raters <- sbr |>
  select(2:5)

ggpairs(raters)
```

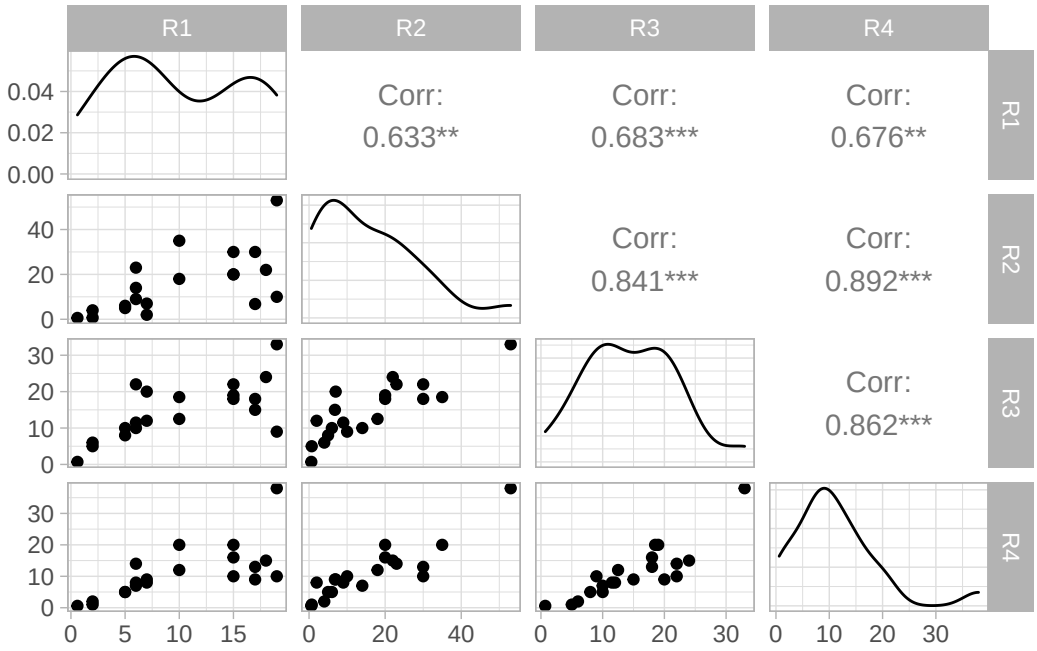


Figure 4.6: Correlation plots relating severity estimates for all pairs of raters

4.1.1.1 Coefficient of determination

We noticed earlier that the correlation coefficients varied across all pairs of rater. Sometimes, the means of squared Pearson's R values (R2), or the coefficient of determination is used as a measure of inter-rater reliability. We can further examine the pair-wise correlations in more details using the `correlation` function of the *performance* package.

```
library(correlation)
raters_cor <- correlation(raters)
library(knitr)
kable(raters_cor[1:3]) # only first 3 variables
```

Table 4.1: Pearson correlation coefficients for all pairs of raters

	Parameter1	Parameter2	r
R1	R2	0.6325037
R1	R3	0.6825936
R1	R4	0.6756986
R2	R3	0.8413333
R2	R4	0.8922181

	Parameter1	Parameter2	r
R3		R4	0.8615470

The means of coefficient of determination can be easily obtained as follows.

```
# All pairwise R2
round(raters_cor$r^2,2)
```

```
[1] 0.40 0.47 0.46 0.71 0.80 0.74
```

```
# means of R2
round(mean(raters_cor$r^2), 2)
```

```
[1] 0.59
```

4.1.1.2 Intraclass Correlation Coefficient

A common statistic to report in reliability studies is the Intraclass Correlation Coefficient (ICC). There are several formulations for the ICC whose choice depend on the particular experimental design. Following the convention of the seminal work by Shrout and Fleiss (1979), there are three main ICCs:

- One-way random effects model, ICC(1,1): in our context, each leaf is rated by different raters who are considered as sampled from a larger pool of raters (random effects)
- Two-way random effects model, ICC(2,1): both raters and leaves are viewed as random effects
- Two-way mixed model, ICC(3,1): raters are considered as fixed effects and leaves are considered as random.

Additionally, the ICC may depend on whether the ratings are an average or not of several ratings. When an average is considered, these are called ICC(1,k), ICC(2,k) and ICC(3,k).

The ICC can be computed using the `ICC()` or the `icc()` functions of the *psych* or *irr* packages, respectively. They both provide the coefficient, F value, and the upper and lower bounds of the 95% confidence interval.

```
library(psych)
ic <- ICC(raters)
```

```
kable(ic$results[1:2]) # only selected columns
```

	type	ICC
Single_raters_absolute	ICC1	0.6405024
Single_random_raters	ICC2	0.6464122
Single_fixed_raters	ICC3	0.6919099
Average_raters_absolute	ICC1k	0.8769479
Average_random_raters	ICC2k	0.8797008
Average_fixed_raters	ICC3k	0.8998319

```
# call ic list for full results
```

The output of interest is a dataframe with the results of all distinct ICCs. We note that the ICC1 and ICC2 gave very close results. Now, let's obtain the various ICCs using the *irr* package. Differently from the the `ICC()` function, this one requires further specification of the model to use.

```
library(irr)
icc(raters, "oneway")
```

```
Single Score Intraclass Correlation

  Model: oneway
  Type : consistency

  Subjects = 20
    Raters = 4
    ICC(1) = 0.641

F-Test, H0: r0 = 0 ; H1: r0 > 0
  F(19,60) = 8.13 , p = 1.8e-10

95%-Confidence Interval for ICC Population Values:
 0.44 < ICC < 0.813
```

```
# The one used in the SBR paper
icc(raters, "twoway")
```

```
Single Score Intraclass Correlation

  Model: twoway
  Type : consistency

  Subjects = 20
    Raters = 4
  ICC(C,1) = 0.692

F-Test, H0: r0 = 0 ; H1: r0 > 0
  F(19,57) = 9.98 , p = 6.08e-12

95%-Confidence Interval for ICC Population Values:
 0.503 < ICC < 0.845
```

4.1.1.3 Overall Concordance Correlation Coefficient

Another useful index is the Overall Concordance Correlation Coefficient (OCCC) for evaluating agreement among multiple observers. It was proposed by Barnhart et al. (2002) based on the original index proposed by Lin (1989), earlier defined in the context of two fixed observers. In the paper, the authors introduced the OCCC in terms of the interobserver variability for assessing agreement among multiple fixed observers. As outcome, and similar to the original CCC, the approach addresses the precision and accuracy indices as components of the OCCC. The `epi.occc` function of the *epiR* packge does the job but it does compute a confidence interval.

```
library(epiR)
epi.occc(raters, na.rm = FALSE, pairs = TRUE)
```

```
Overall CCC           0.6372
Overall precision     0.7843
Overall accuracy      0.8125
```

4.1.2 Intrarater reliability

As defined, the intrarater reliability is also known as repeatability, because it measures consistency by the same rater at repeated assessments (e.g. different times) on the same sample. In some studies, we may be interested in testing whether a new method increases repeatability of assessments by a single rater compared with another one. The same indices used for assessing

reproducibility (interrater) can be used to assess repeatability, and these are reported at the rater level.

4.1.3 Precision

When assessing precision, one measures the variability of the estimates (or measurements) of disease on the same sampling units obtained by different raters (or instruments). A very high precision does not mean that the estimates are closer to the actual value (which is given by measures of bias). However, precision is a component of overall accuracy, or agreement. It is given by the Pearson's correlation coefficient.

Different from reliability, that requires only the estimates or measures by the raters, now we need a reference (gold standard) value to compare the estimates to. These can be an accurate rater or measures by an instrument. Let's get back to the soybean rust severity estimation dataset and add a column for the (assumed) actual values of severity on each leaf. In that work, the actual severity values were obtained using image analysis.

```
sbr <- tibble::tribble(
~leaf, ~actual, ~R1, ~R2,  ~R3, ~R4,
1,    0.25, 0.6, 0.6,  0.7, 0.6,
2,     2.5,   2, 0.7,    5,   1,
3,    7.24,   5,   5,    8,   5,
4,    7.31,   2,   4,    6,   2,
5,    9.07,   6,  14,   10,   7,
6,    11.6,   5,   6,   10,   5,
7,   12.46,  10,  18, 12.5,  12,
8,    13.1,  15,  30,   22,  10,
9,   14.61,   7,   2,   12,   8,
10,  16.06,   6,   9, 11.5,   8,
11,   16.7,   7,   7,   20,   9,
12,   19.5,   6,  23,   22,  14,
13,  20.75,  10,  35, 18.5,  20,
14,  23.56,  19,  10,    9,  10,
15,  23.77,  15,  20,   19,  20,
16,  24.45,  17,  30,   18,  13,
17,  25.78,  19,  53,   33,  38,
18,  26.03,  17, 6.8,   15,   9,
19,  26.42,  15,  20,   18,  16,
20,  28.89,  18,  22,   24,  15
        )
```

We can explore visually via scatter plots the relationships between the actual value and the estimates by each rater (Figure **??**). To facilitate, we need the data in the long format.

```r
sbr2 <- sbr |>
  pivot_longer(3:6, names_to = "rater",
               values_to = "estimate")

sbr2 |>
  ggplot(aes(actual, estimate))+
  geom_point(size = 2, alpha = 0.7)+
  facet_wrap(~rater)+
  ylim(0,45)+
  xlim(0,45)+
  geom_abline(intercept = 0, slope =1)+
  labs(x = "Actual severity (%)",
       y = "Estimate severity (%)")
```

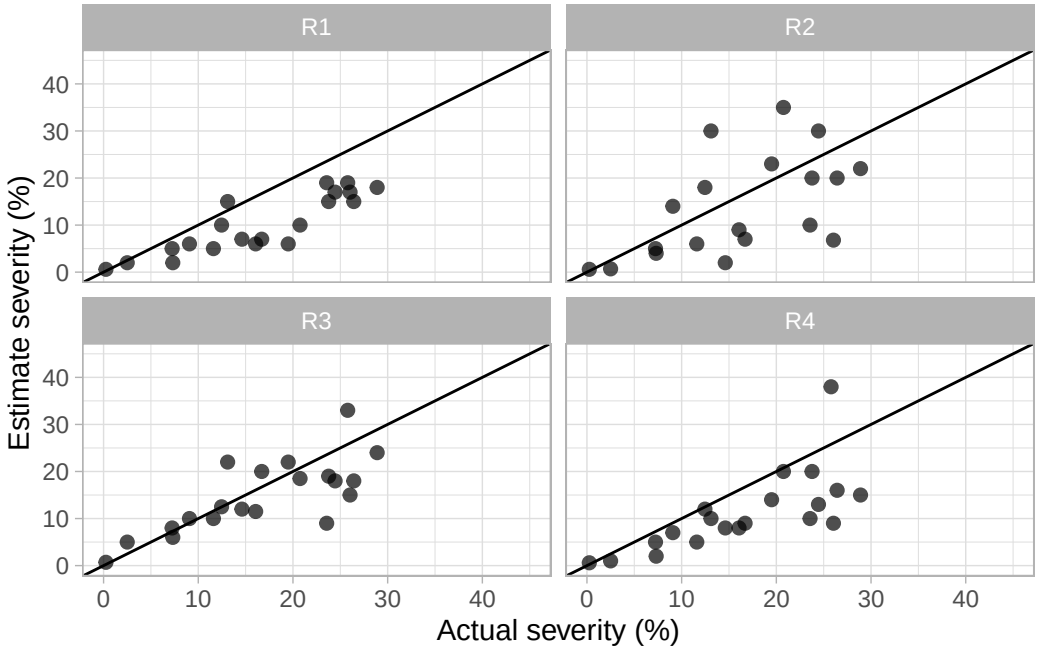


Figure 4.7: Scatterplots for the relationship between estimated and actual severity for each rater

The Pearson's r for the relationship, or the precision of the estimates by each rater, can be obtained using the `correlation` function of the *correlation* package.

```
precision <- sbr2 |>
  select(-leaf) |>
  group_by(rater) |>
  correlation()

kable(precision[1:4])
```

	Group	Parameter1	Parameter2	r
R1	actual	estimate	0.8725643
R2	actual	estimate	0.5845291
R3	actual	estimate	0.7531983
R4	actual	estimate	0.7108260

The mean precision can then be obtained.

```
mean(precision$r)
```

```
[1] 0.7302795
```

4.1.4 Accuracy

4.1.4.1 Absolute errors

It is useful to visualize the errors of the estimates which are obtained by subtracting the estimates from the actual severity values. This plot allows to visualize patterns in over or underestimations across a range of actual severity values.

```
sbr2 |>
  ggplot(aes(actual, estimate-actual))+
  geom_point(size = 3, alpha = 0.7)+
  facet_wrap(~rater)+
  geom_hline(yintercept = 0)+
  labs(x = "Actual severity (%)",
       y = "Error (Estimate - Actual)")
```

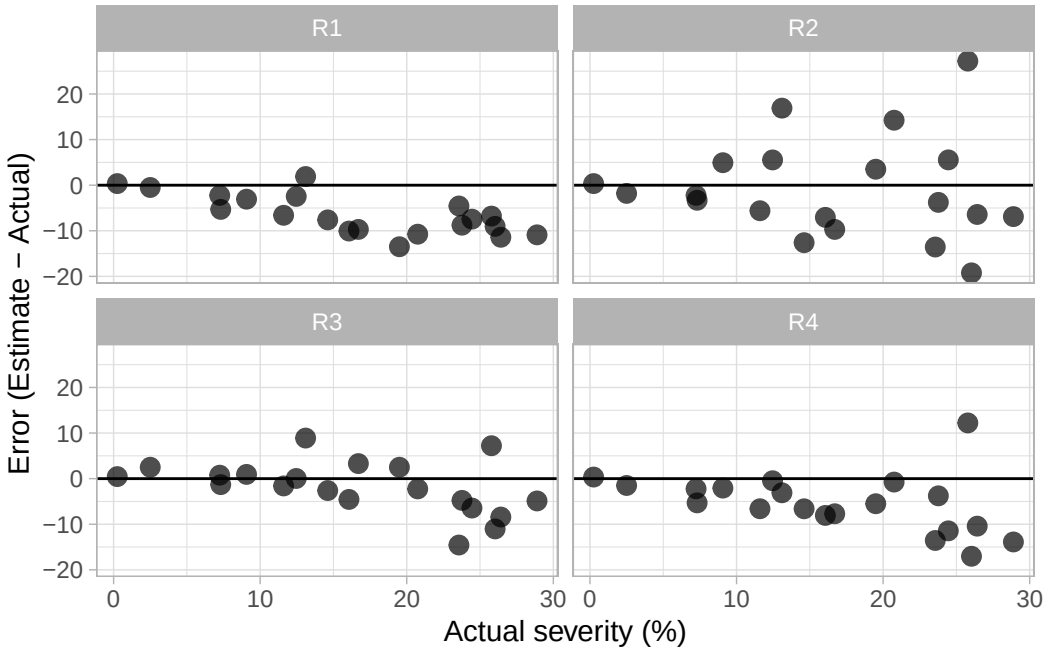


Figure 4.8: Error (estimated - actual) of visual severity estimates

4.1.4.2 Concordance correlation coefficient

Lin's (1989, 2000) proposed the concordance correlation coefficient (CCC) for agreement on a continuous measure obtained by two methods. The CCC combines measures of both precision and accuracy to determine how far the observed data deviate from the line of perfect concordance. Lin's CCC increases in value as a function of the nearness of the data reduced major axis to the line of perfect concordance (the accuracy of the data) and of the tightness of the data about its reduced major axis (the precision of the data).

The `epi.ccc` function of the *epiR* package allows to obtain the Lin's CCC statistics. Let's filter only rater 2 and calculate the CCC statistics for this rater.

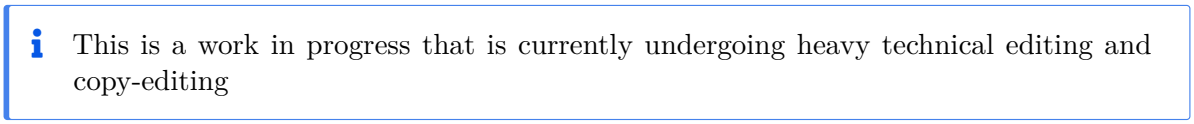
```
library(epiR)
# Only rater 2
sbr3 <- sbr2 |> filter(rater == "R2")
ccc <- epi.ccc(sbr3$actual, sbr3$estimate)
# Concordance coefficient
rho <- ccc$rho.c[,1]
# Bias coefficient
Cb <- ccc$C.b
```

```r
# Precision
r <- ccc$C.b*ccc$rho.c[,1]
# Scale-shift
ss <- ccc$s.shift
# Location-shift
ls <- ccc$l.shift
Metrics <- c("Agreement", "Bias coefficient", "Precision", "scale-shift", "location-shift"
Value <- c(rho, Cb, r, ss, ls)
res <- data.frame(Metrics, Value)
kable(res)
```

Table 4.4: Statitics of the concordance correlation coefficient summarizing accuracy and precision of visual severity estimates of soybean rust for four raters

Metrics	Value
Agreement	0.5230656
Bias coefficient	0.8948494
Precision	0.4680649
scale-shift	1.6091178
location-shift	-0.0666069

5 Measuring severity



Among the different ways to express plant disease severity, the percent area affected (symptomatic) by the disease is one of the most common especially when dealing with diseases that affect leaves. To evaluate whether the visual estimates of plant disease severity are sufficiently accurate (as seen in the previous chapter), one needs the actual severity values. These are also needed when preparing standard area diagrams (SADs) which are diagrammatic representations of severity values used as an aid prior or during the visual assessment to standardize and produce more accurate results across different raters (Del Ponte et al. 2017).

The actual severity values are usually approximated using image analysis where the image is segmented and each pixel of the image is labeled according to three different classes:

1. Diseased (or symptomatic)
2. Non-diseased (or healthy)
3. Background (the non-plant portion of the image)

The ratio between the diseased area and total area of the unit (e.g. the whole plant organ or section image) gives the proportion of diseased area or the percent area affected (when multiplied by 100). Several different proprietary or open-source software has been used by researchers to determine the actual severity according to a review on standard area diagrams (Del Ponte et al. 2017).

Here, we will use the `measure_disease` function of the *pliman* (Plant IMage ANalysis) (Olivoto 2022) R package to obtain measures of severity. The package was compared with other software for determining plant disease severity on five different plant diseases and showed to produce concordant results for most of the cases (Olivoto et al. 2022).

There are basically two methods to measure severity. The first is based on image palettes that define each class of the image. The second is based on RGB-based indices (Alves et al. 2021).

5.1 Image palettes

The most critical is the initial step, when the user needs to correctly define the color palettes. In *pliman* the palettes are actually separate images representing each of three classes named background (b), symptomatic (s) and healthy (h).

The reference image palettes can be constructed by manually sampling small areas of the image and producing a composite image. Of course, the results may vary depending on how these areas are chosen. A work on the validation of the *pliman* to determine disease severity showed the effect of different palettes prepared independently by three researchers (Olivoto et al. 2022). The observation of the processed masks during the calibration of the palettes is important to create reference palettes that are most representative of the respective class.

Here, I cut and pasted several sections of images representative of each class from a few leaves into a Google slide. Once the image palette was ready, I exported each one as a separate image PNG file (JPG also works). These were named: sbr_b.png, sbr_h.png and sbr_s.png. They can be found here in this folder for downloading.

Now that we have the image palettes, we can start by importing them into the environment for further analysis. Let's create an image object for each palette named h (healthy), s (symptoms) and b (background).

```
library(pliman)
h <- image_import("imgs/sbr_h.png")
s <- image_import("imgs/sbr_s.png")
b <- image_import("imgs/sbr_b.png")
```

We can visualize the imported image palettes using the `image_combine()` function.

```
image_combine(h, s, b, ncol =3)
```

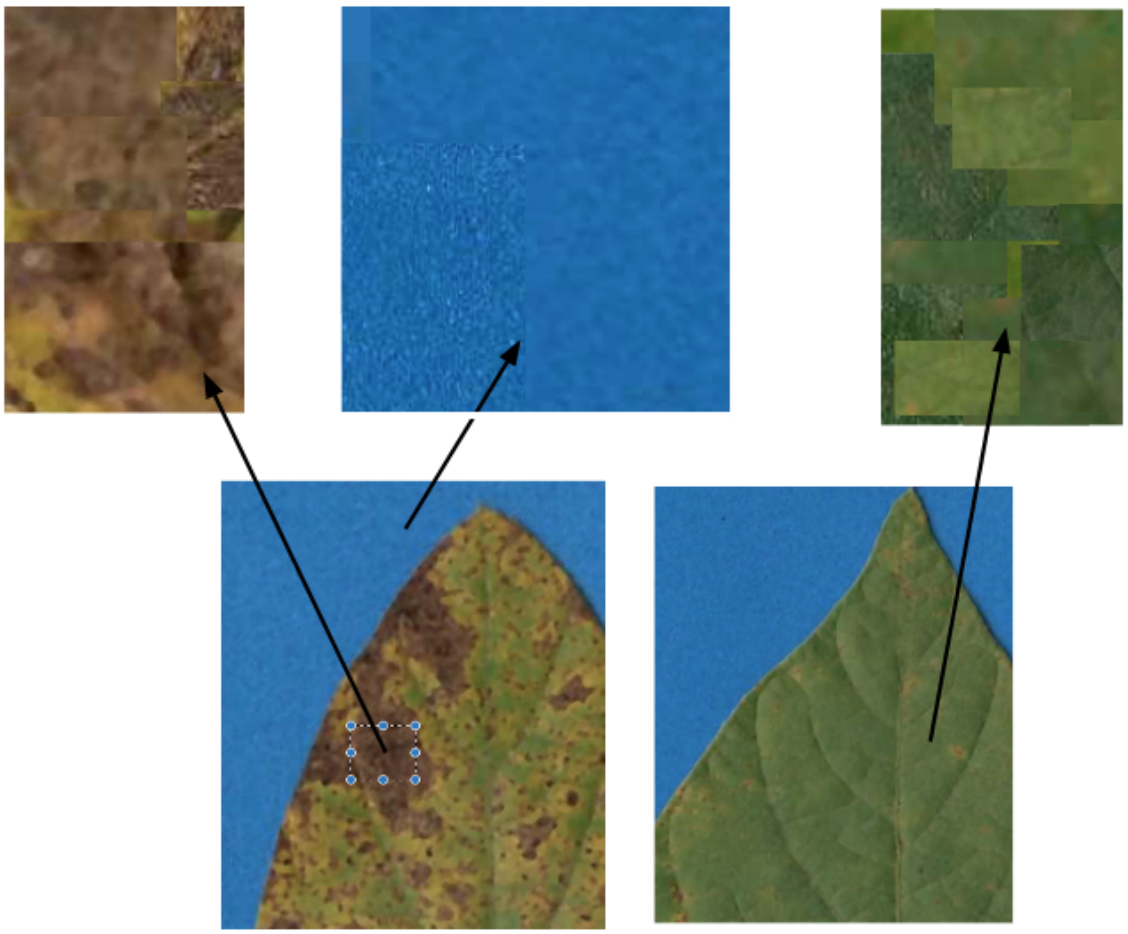


Figure 5.1: Preparation of image palettes by manually sampling fraction of the images that represent background, heatlhy leaf and lesions

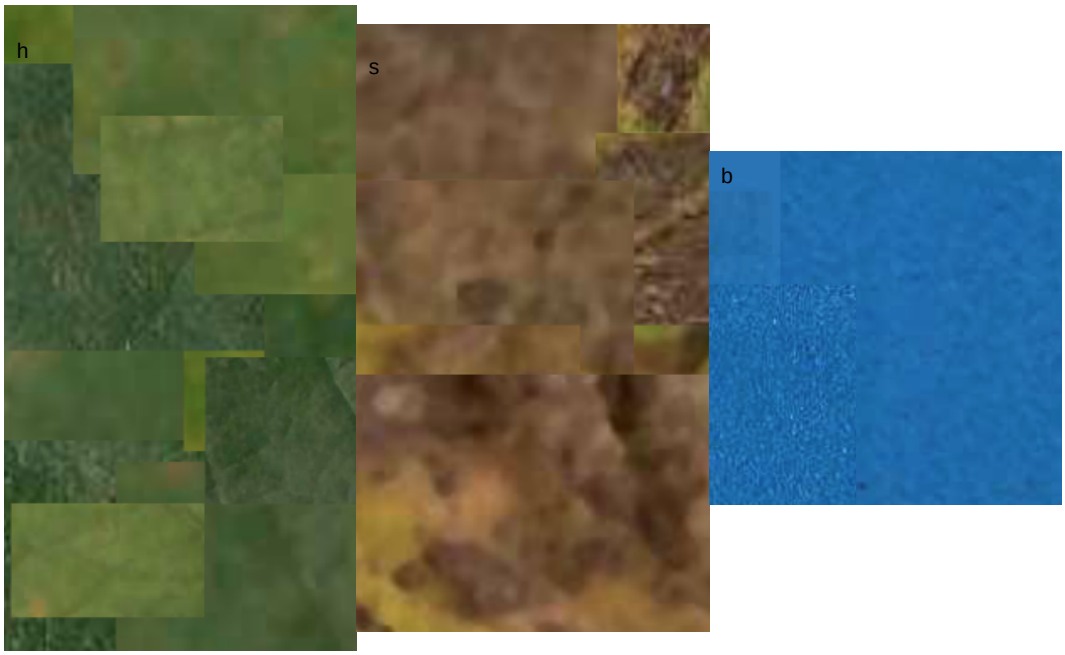


Figure 5.2: Image palettes created to segment images into background, sypomtoms and healthy area of the image

5.2 Measuring severity

5.2.1 Single image

To determine severity in a single image (img46.png), the image file needs to be loaded and assigned to an object using the same `image_import()` function used to load the palettes. We can then visualize the image, again using `image_combine()`.

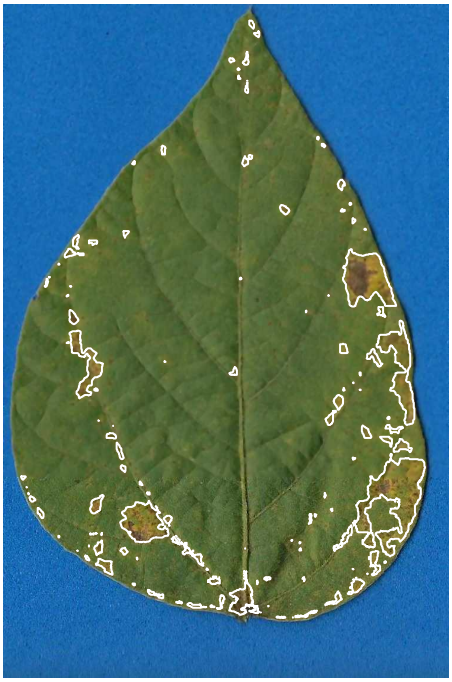
> **Tip**
>
> The collection of images used in this chapter can be found here.

```
img <- image_import("imgs/originals/img46.png")
image_combine(img)
```

Figure 5.3: Imported image for further analysis

Now the fun begins with the `measure_disease()` function to determine severity. Four arguments are needed when using the reference image palettes, the one representing the target image and each of the three images of the color palettes. As the author of the package says "pliman will take care of all details!". The severity is the value shown under symptomatic in the output.

```
set.seed(123)
measure_disease(
  img = img,
  img_healthy = h,
  img_symptoms = s,
  img_background = b,
  show_image = TRUE
)
```



```
$severity
   healthy symptomatic
1 92.68302    7.316983

$shape
NULL

$statistics
NULL

attr(,"class")
[1] "plm_disease"
```

5.2.2 Multiple images

Measuring severity in single images is fun, but usually we don't have a single image to process but several. It would take a longer time to process each one using the above procedure, thus becoming tedious.

To automate the process, *pliman* offers a batch processing approach. For such, instead of using `img` argument, one can use `pattern` and define the prefix of names of the images. In addition, we also need to define the folder where the original files are located.

If the users wants to save the processed masks, the `save_image` argument needs to be set to TRUE and the directory where the images will be saved also should be informed. Check below how to process 10 images of soybean rust symptoms. The outcome is a `list` object with the measures of the percent healthy and percent symptomatic area for each leaf in the `severity` object.

```
pliman <- measure_disease(
  pattern = "img",
  dir_original = "imgs/originals" ,
  dir_processed = "imgs/processed",
  save_image = TRUE,
  img_healthy = h,
  img_symptoms = s,
  img_background = b,
  show_image = FALSE
)
```

```
Processing image img11 |====                                              | 10% 00:00:00

Processing image img35 |========                                          | 20% 00:00:02

Processing image img37 |============                                      | 30% 00:00:03

Processing image img38 |================                                  | 40% 00:00:03

Processing image img46 |====================                              | 50% 00:00:04

Processing image img5 |========================                          | 60% 00:00:05

Processing image img63 |============================                      | 70% 00:00:07

Processing image img67 |================================                  | 80% 00:00:09

Processing image img70 |====================================              | 90% 00:00:11

Processing image img75 |========================================|| 100% 00:00:12
```

```
severity <- pliman$severity
severity
```

```
     img  healthy symptomatic
1  img11 70.80072  29.1992835
2  img35 46.96430  53.0357002
3  img37 60.49390  39.5060986
4  img38 79.14737  20.8526306
5  img46 93.15143   6.8485680
6   img5 20.53977  79.4602312
7  img63 97.15698   2.8430190
8  img67 99.83723   0.1627709
9  img70 35.58683  64.4131683
10 img75 93.04517   6.9548329
```

With the argument `save_image` set to TRUE, the images are all saved in the folder with the standard prefix "proc."

Let's have a look at one of the processed images.

5.3 How good are these measurements?

These 10 images were previously processed in QUANT software for measuring severity which is also based on image threshold. Let's create a tibble for the image code and respective "actual" severity - assuming QUANT measures as reference.

```
library(tidyverse)
quant <- tribble(
  ~img, ~actual,
   "img5",     75,
  "img11",     24,
  "img35",     52,
  "img37",     38,
  "img38",     17,
  "img46",      7,
  "img63",    2.5,
  "img67",   0.25,
  "img70",     67,
  "img75",     10
  )
```

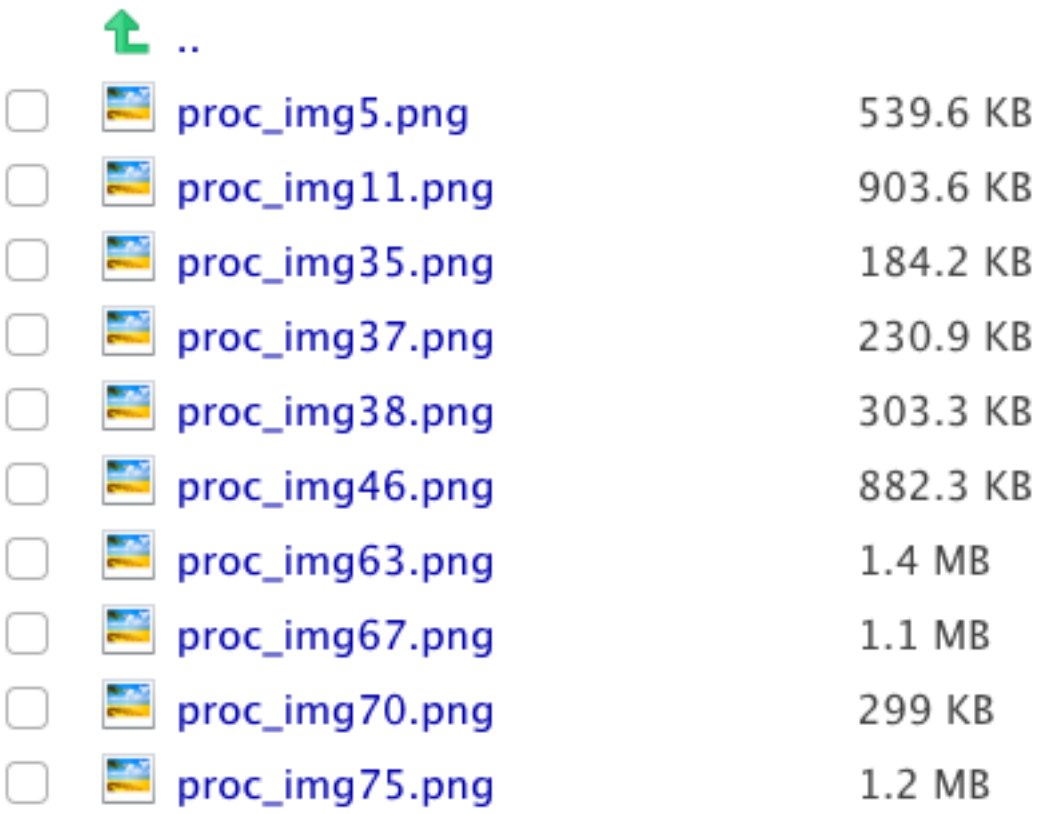


Figure 5.4: Images created by pliman and exported to a specific folder

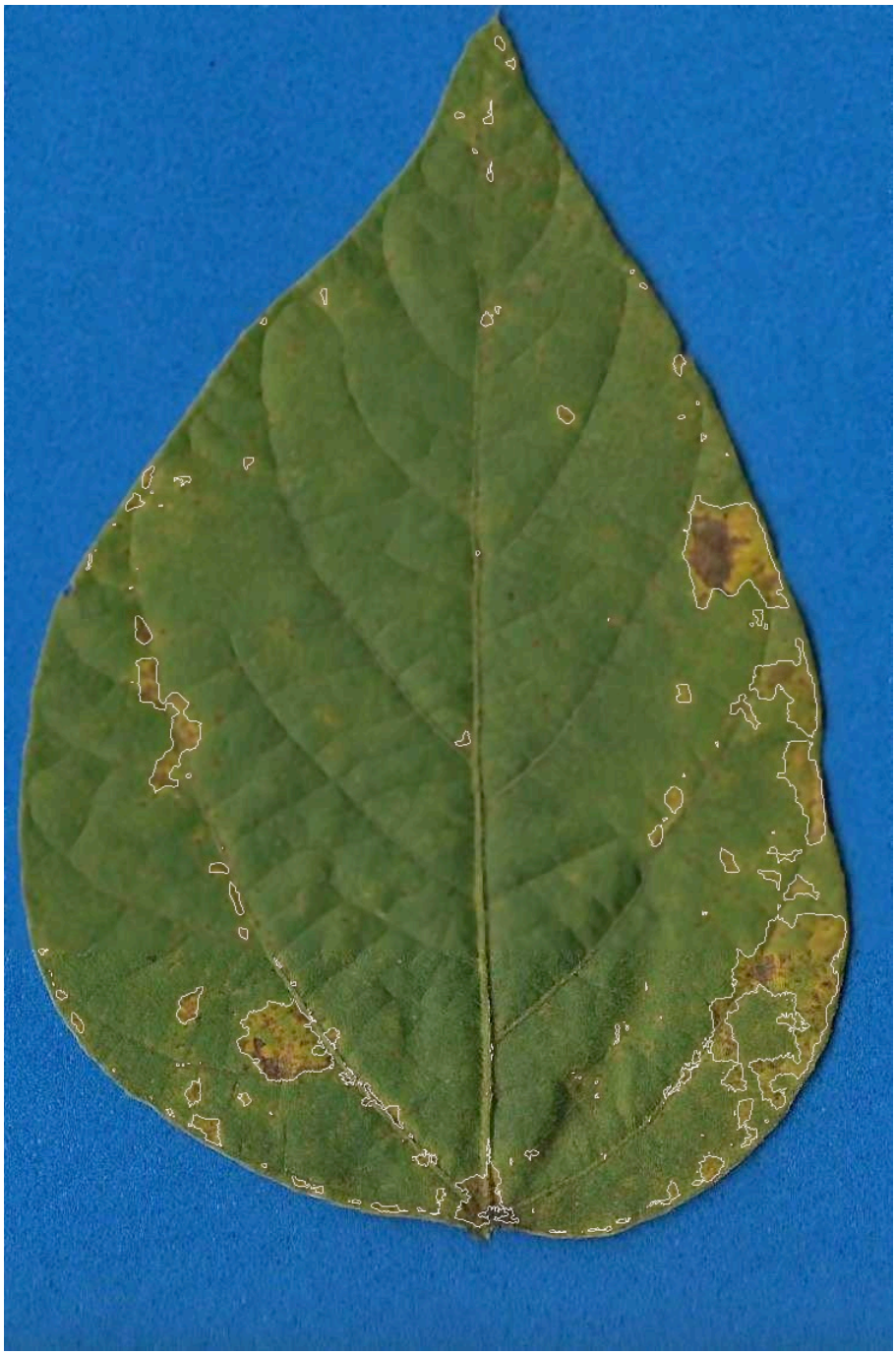


Figure 5.5: Figure created by pliman after batch processing to segment the images and calculate percent area covered by symptoms. The symptomatic area is delinated in the image.

We can now combine the two dataframes and produce a scatter plot relating the two measures.

```
dat <- left_join(severity, quant)
```

```
Joining, by = "img"
```

```
dat %>%
  ggplot(aes(actual, symptomatic)) +
  geom_point(size = 5, shape = 16, color = "gray50") +
  ylim(0, 100) +
  xlim(0, 100) +
  geom_abline(slope = 1, intercept = 0) +
  theme_light() +
  labs(x = "Quant",
       y = "pliman")
```

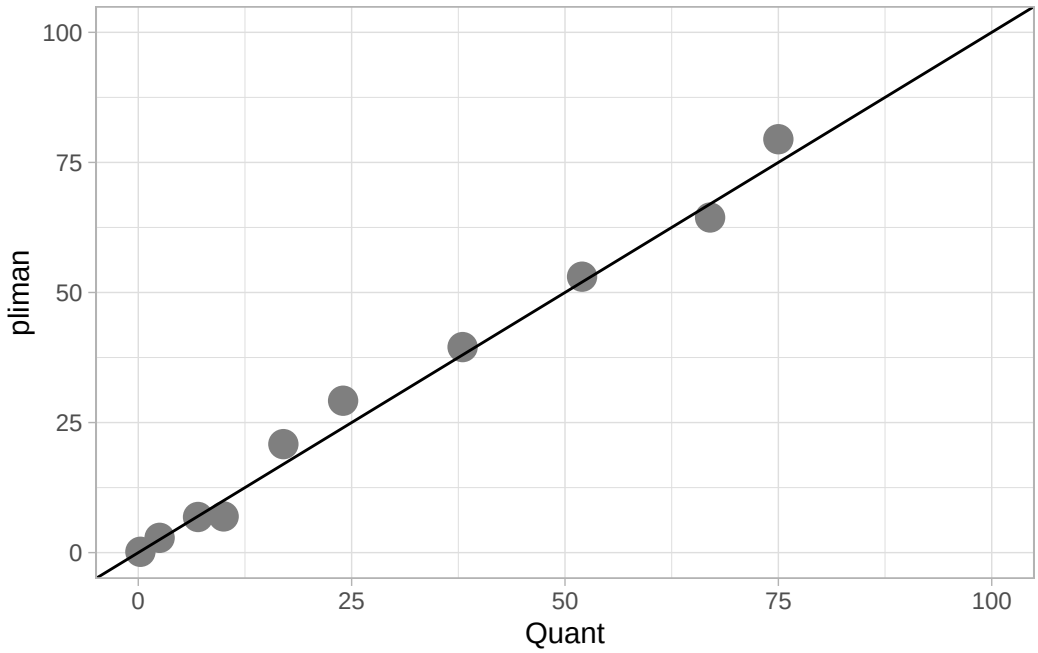


Figure 5.6: Scatter plot for the relationship between severity values measured by pliman and Quant software

The concordance correlation coefficient is a test for agreement between two observers or two methods (see previous chapter). It is an indication of how accurate the *pliman* measures are

compared with a standard. The coefficient is greater than 0.99 (1.0 is perfect concordance), suggesting an excellent agreement!

```
library(epiR)
ccc <- epi.ccc(dat$actual, dat$symptomatic)
ccc$rho.c
```

```
       est     lower     upper
1 0.9940941 0.9774812 0.9984606
```

In conclusion, the most critical step, as mentioned, is the definition of the reference image palettes. A few preliminary runs may be needed for a few images to check whether the segmentation is being performed correctly, based on visual judgement. This is no different than any other color-threshold based methods when the choices made by the user affect the final result and contribute to variation among assessors. The cons are the same encountered in the direct competitors, which is the necessity to have images obtained at uniform and controlled conditions, especially a contrasting background.

Part II

Temporal analysis

6 Disease progress curves

> ℹ This is a work in progress that is currently undergoing heavy technical editing and copy-editing

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 percentage (0 to 100%).

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

```
library(tidyverse) # essential packages
library(cowplot) # for themes
theme_set(theme_bw(base_size = 16)) # set global theme
```

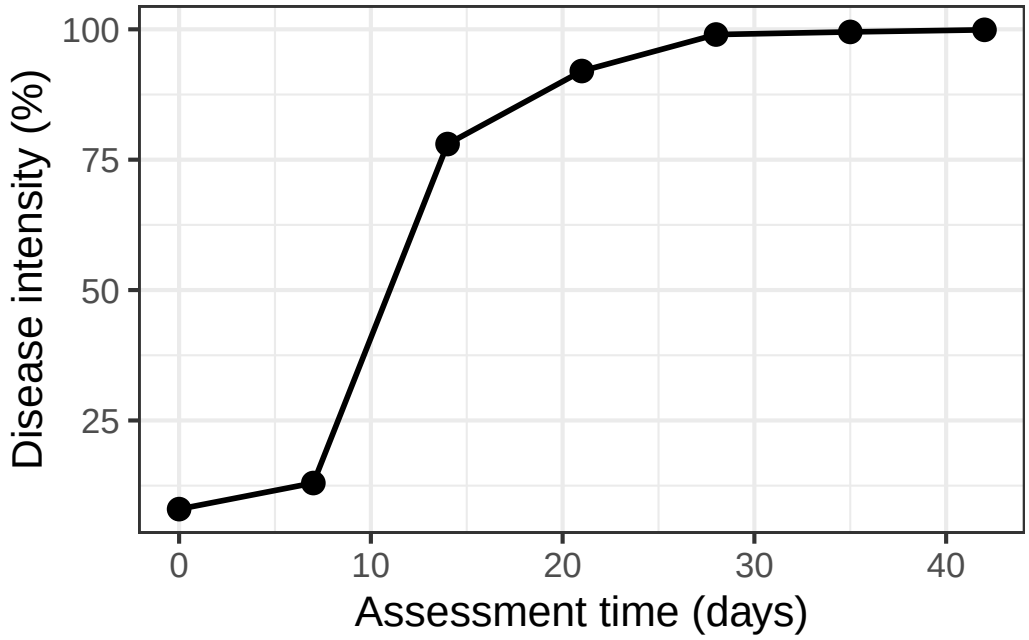
There are several ways to create a dataframe in R. I like to use the `tribble` function as below. The entered data will be assigned to a dataframe called `dpc`.

```
dpc <-
  tribble(
   ~t,  ~y,
   0,  8,
   7,  13,
  14,  78,
  21,  92,
  28,  99,
  35, 99.5,
```

```
  42, 99.9,
  )
```

Now the plot

```
dpc |>
  ggplot(aes(t, y)) +
  geom_line(size = 1)+
  geom_point(size = 4, shape = 16)+
  labs(x = "Assessment time (days)",
       y = "Disease intensity (%)")
```



6.1 Curve descriptors: AUDPC

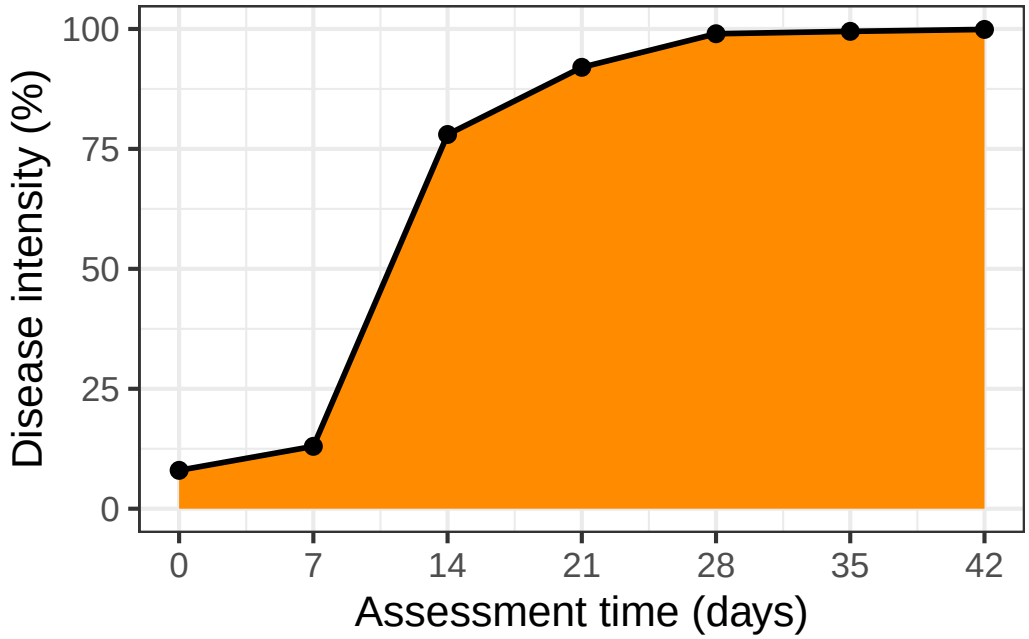
The depiction and analysis of disease progress curves can provide useful information for gaining understanding of the underlying epidemic process. The curves are extensively used to evaluate how disease control measures affect epidemics. When characterizing DPCs, a researcher may be interested in describing 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

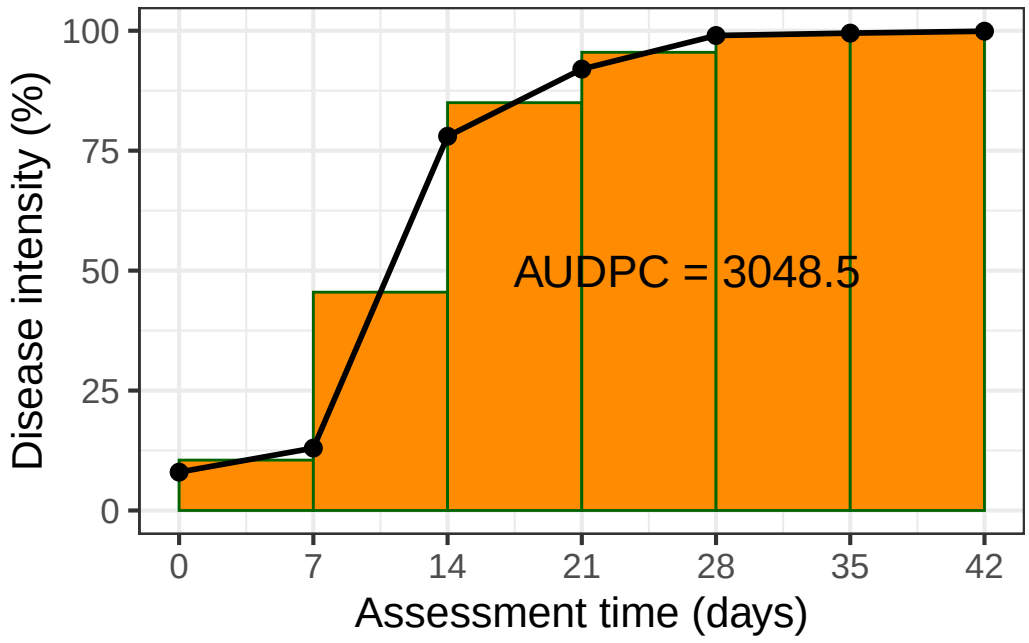
the study. In general, the goal is to synthesize similarities and different among epidemics based on common descriptors of the disease progress curves. For example, the simple appraisal of the disease intensity at any time during the course of the epidemic should be sufficient for certain situations. 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.

```
library(ggthemes)
dpc1 <- dpc |>
  ggplot(aes(t, y)) +
  labs(x = "Assessment time (days)",
       y = "Disease intensity (%)")+
    geom_area(fill = "darkorange")+
    geom_line(size = 1)+
  geom_point(size = 3, shape = 16)+
  scale_x_continuous(breaks = c(0, 7, 14, 21, 28, 35, 42))

dpc1
```



```r
dpc2 <- dpc |>
  ggplot(aes(t, y)) +
  labs(x = "Assessment time (days)",
       y = "Disease intensity (%)")+
  annotate("rect", xmin = dpc$t[1], xmax = dpc$t[2],
           ymin = 0, ymax = (dpc$y[1]+ dpc$y[2])/2,
           color = "darkgreen", fill = "darkorange")+
   annotate("rect", xmin = dpc$t[2], xmax = dpc$t[3],
            ymin = 0, ymax = (dpc$y[2]+ dpc$y[3])/2,
            color = "darkgreen", fill = "darkorange")+
   annotate("rect", xmin = dpc$t[3], xmax = dpc$t[4],
            ymin = 0, ymax = (dpc$y[3]+ dpc$y[4])/2,
            color = "darkgreen",, fill = "darkorange")+
   annotate("rect", xmin = dpc$t[4], xmax = dpc$t[5],
            ymin = 0, ymax = (dpc$y[4]+ dpc$y[5])/2,
            color = "darkgreen", fill = "darkorange")+
   annotate("rect", xmin = dpc$t[5], xmax = dpc$t[6],
            ymin = 0, ymax = (dpc$y[5]+ dpc$y[6])/2,
            color = "darkgreen",fill = "darkorange")+
   annotate("rect", xmin = dpc$t[6], xmax = dpc$t[7],
            ymin = 0, ymax = (dpc$y[6]+ dpc$y[7])/2,
            color = "darkgreen", fill = "darkorange")+
  geom_line(size = 1)+
  geom_point(size = 3, shape = 16)+
  annotate(geom = "text", x = 26.5, y = 50,
           label = "AUDPC = 3048.5", size = 6)+
  scale_x_continuous(breaks = c(0, 7, 14, 21, 28, 35, 42))
dpc2
```



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).

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

```
[1] 3048.15
```

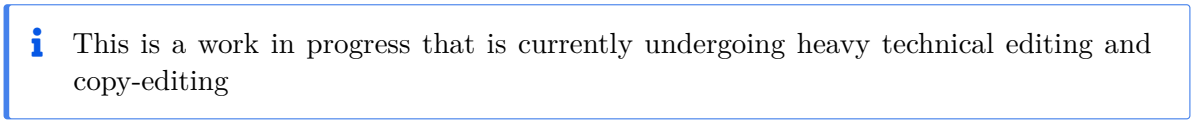
```
# The relative AUDPC
AUDPC(dpc$t, dpc$y,
      y_proportion = FALSE,
      type = "relative")
```

```
[1] 0.72575
```

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

```
[1] 3425.8
```

7 Population models



This is a work in progress that is currently undergoing heavy technical editing and copy-editing

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.

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.

7.0.1 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.

7.0.1.1 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.).

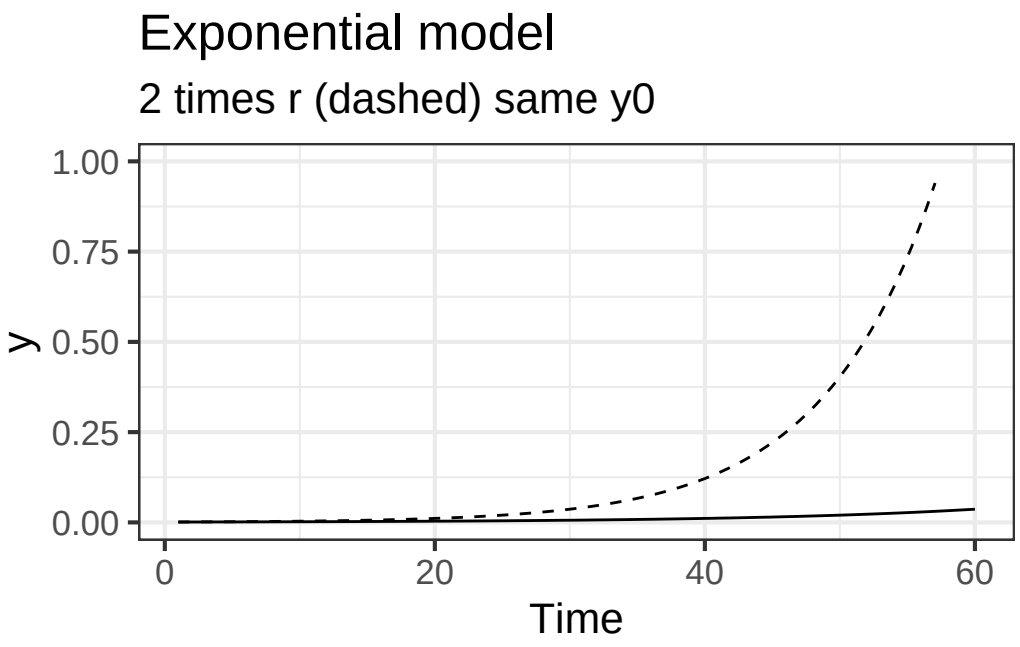
```
library(tidyverse) # essential packages
library(cowplot) # for themes
theme_set(theme_bw(base_size = 16)) # set global theme
```

```
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 ($r = 0.12$)

```
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).
```

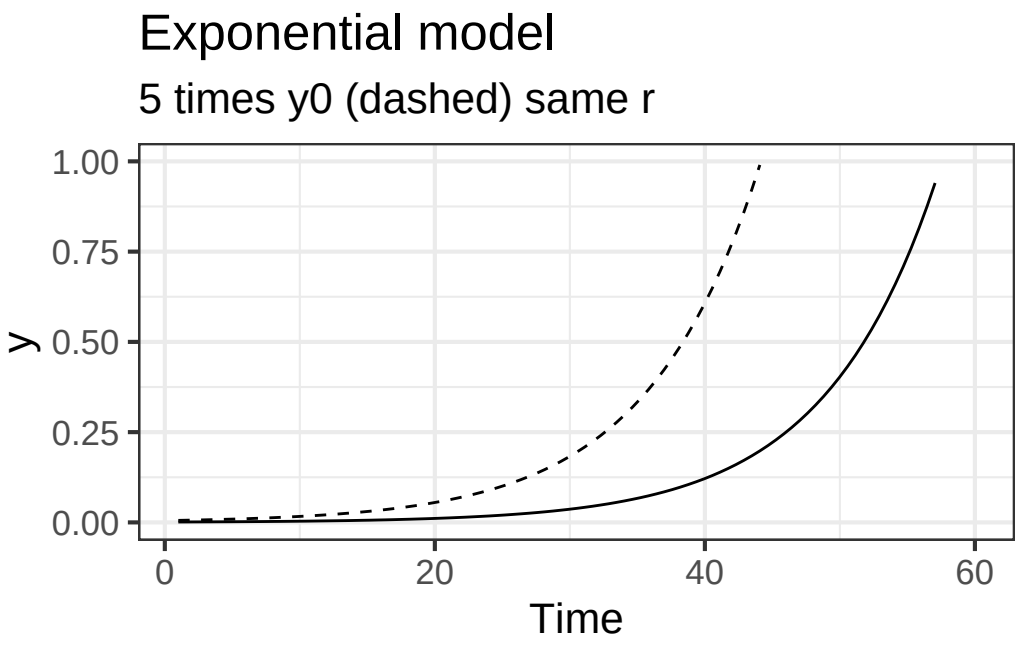


Now the inoculum was increased five times while using the same doubled rate.

```
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).
```



7.0.1.2 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.

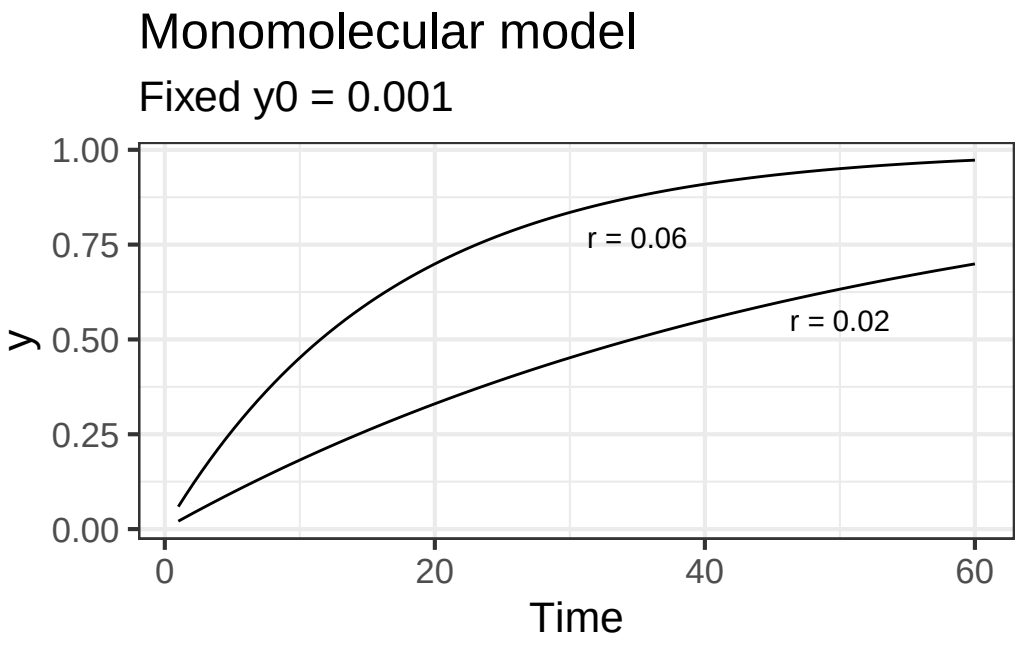
The integral of the monomolecular model is given by

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

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 $(1-y)$.

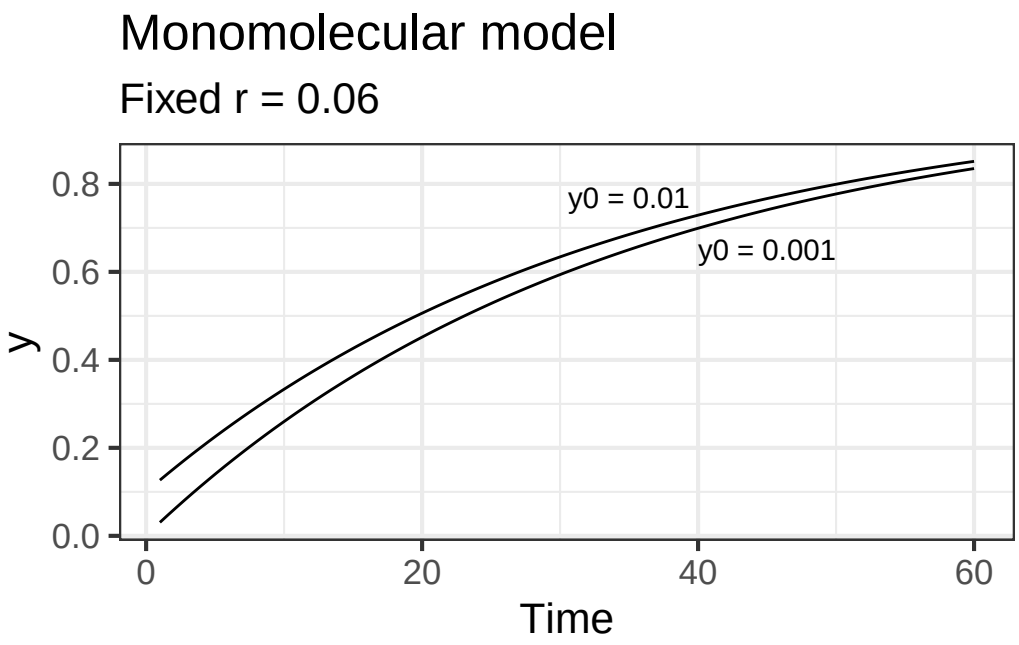
Let's simulate two monomolecular curve with different rate parameters where one is one third of the other.

```
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.

```
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")
```



7.0.1.3 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 $y = 1$.

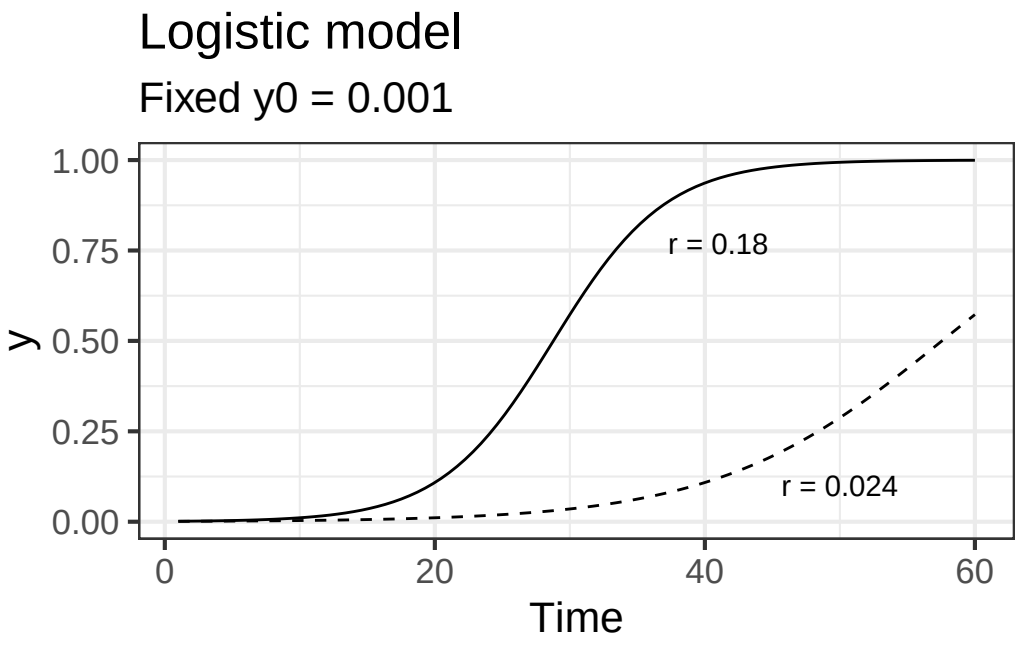
The integral is given by

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

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.

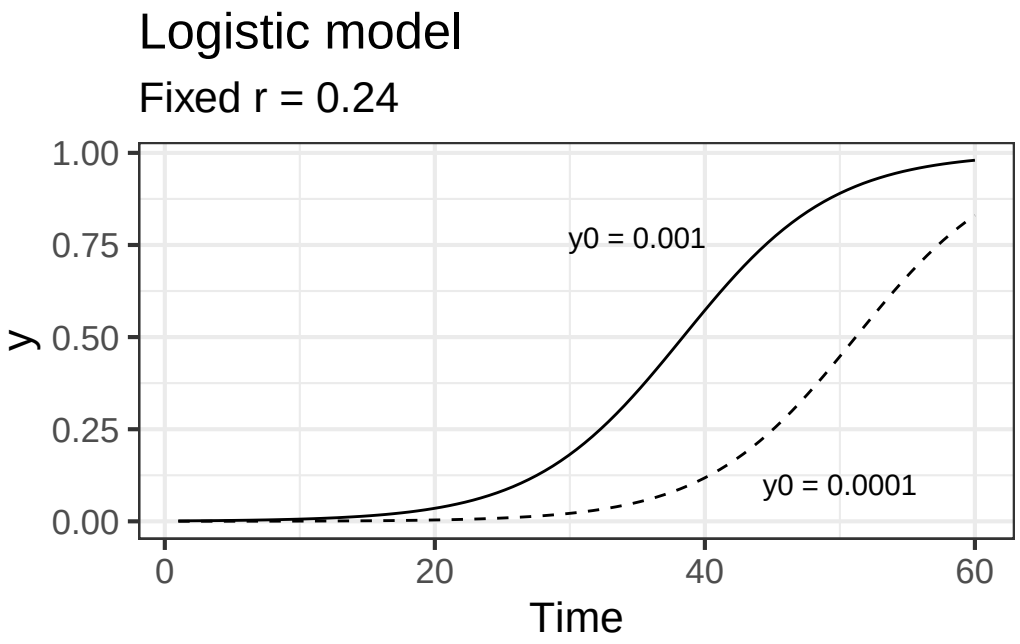
```r
dat |>
  ggplot(aes(t, y)) +
  stat_function(
    linetype = 2,
    fun = function(t) 1 / (1 + ((1 - y0) / y0) * exp(-r * 2 * t))
  ) +
  stat_function(fun = function(t) 1 / (1 + ((1 - y0) / y0) * exp(-r * 4 * t))) +
  labs(title = "Logistic model", subtitle = "Fixed y0 = 0.001", x = "Time") +
  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.

```r
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")
```



7.0.1.4 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

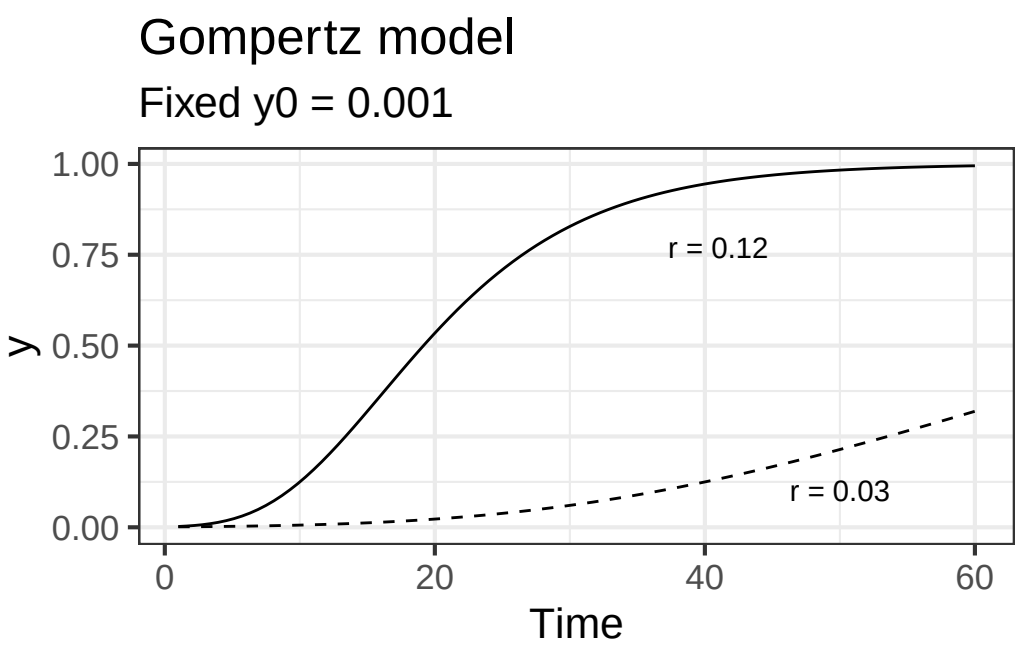
$y = e^{(ln(y0)).e^{-r_G.t)}}$,

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.

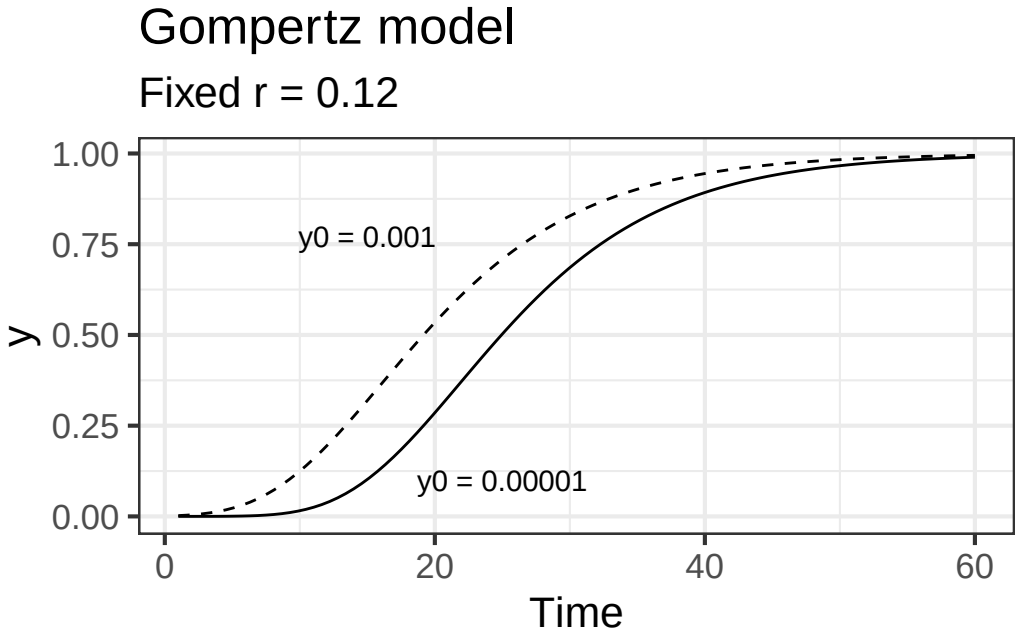
```
dat |>
  ggplot(aes(t, y)) +
  stat_function(
```

```
    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.

```
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")
```


Gompertz model
Fixed r = 0.12
1.00
0.75
0.50
0.25
0.00
y
y0 = 0.001
y0 = 0.00001
0
20
40
60
Time

8 Model fitting

> This is a work in progress that is currently undergoing heavy technical editing and copy-editing

In this tutorial, 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.

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.

8.0.1 Entering data

```
library(tidyverse) # essential packages
library(cowplot) # for themes
theme_set(theme_bw(base_size = 16)) # set global theme
```

```
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,
```

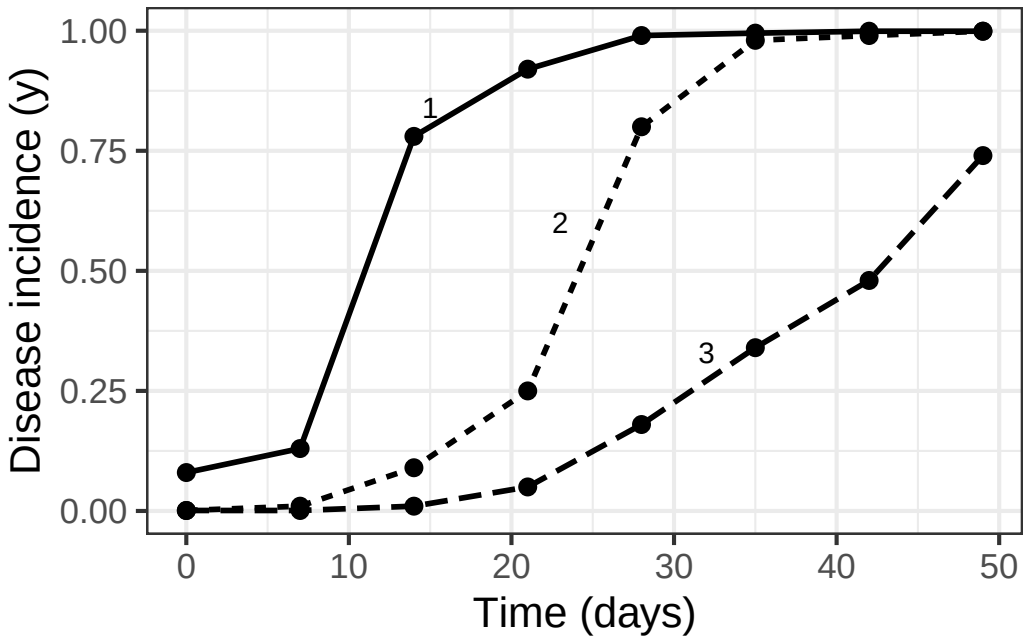
```
  35, 0.995,  0.98,  0.34,
  42, 0.999,  0.99,  0.48,
  49, 0.999, 0.999,  0.74
  )
```

8.0.2 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_line(size = 1)+
  geom_point(size =3, shape = 16)+
  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.

8.0.3 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.

8.0.4 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.

```
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.

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

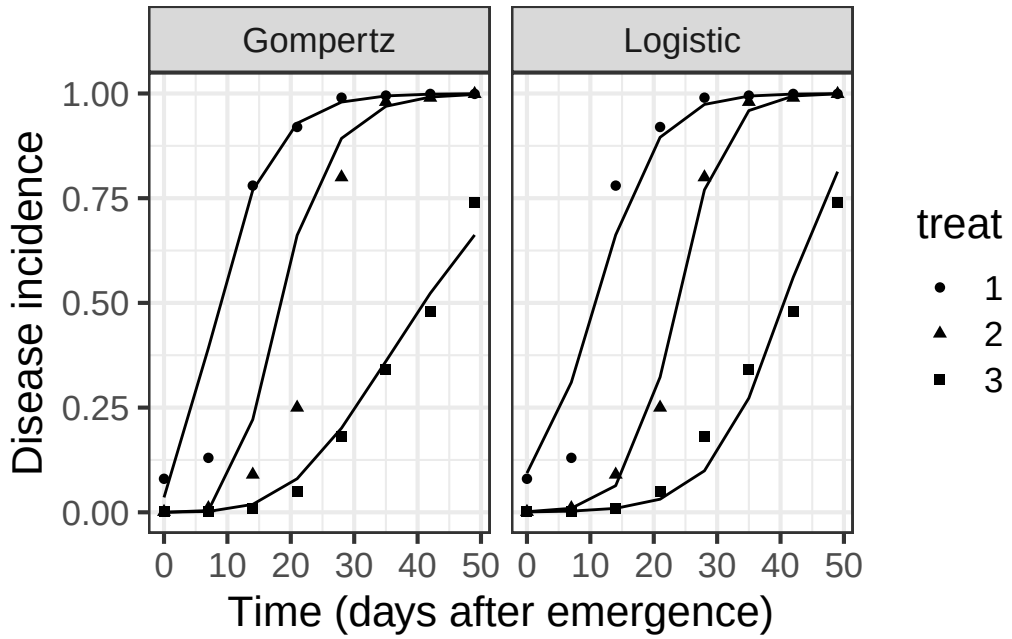
Now let's select the statistics of model fitting. Again, *Epifitter* ranks the models based on the CCC (the higher the better) but it is important to check the RSE as well - the lower the better. In fact, the RSE is more important when the goal is prediction.

```
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.

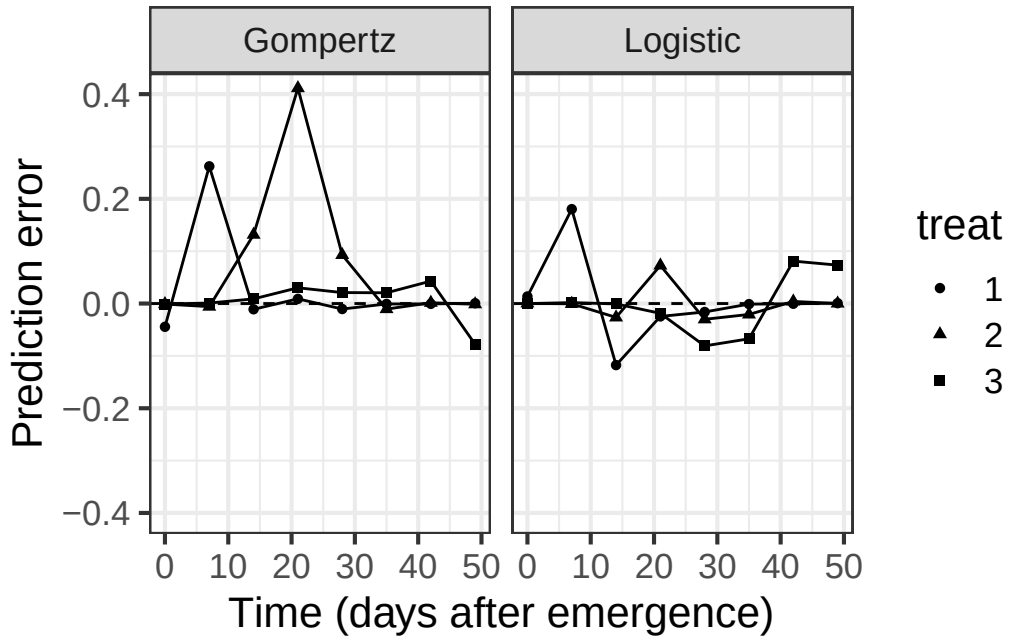
```
epi_all$Data |>
 filter(model %in% c("Gompertz", "Logistic")) |>
  ggplot(aes(time, predicted, shape = treat)) +
  geom_point(aes(time, y)) +
  geom_line() +
  facet_wrap(~ model) +
 coord_cartesian(ylim = c(0, 1)) + # set the max to 0.6
  labs(
    y = "Disease incidence",
    x = "Time (days after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

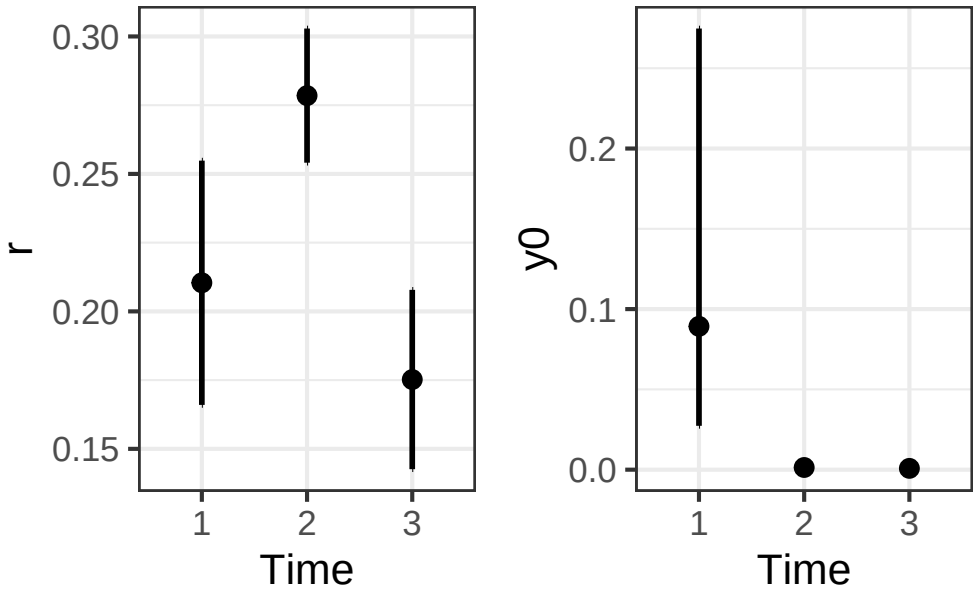
We can produce a plot for visual inference on the differences in the parameters.

```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
```



8.0.5 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`.

8.0.5.1 Loading data

```
budblight <- read_csv("https://raw.githubusercontent.com/emdelponte/epidemiology-R/main/da
```

```
Rows: 64 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): treat
```

```
dbl (3): time, block, y

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

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
```

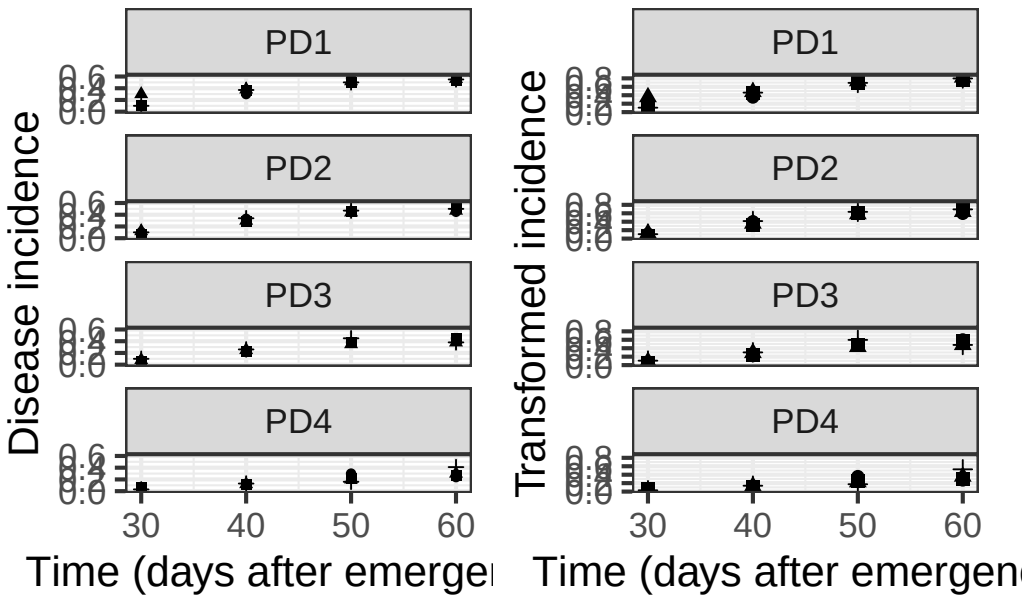
8.0.5.2 Visualizing the DPCs

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)")
```

```
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
```



8.0.5.3 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.

8.0.5.3.1 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
```

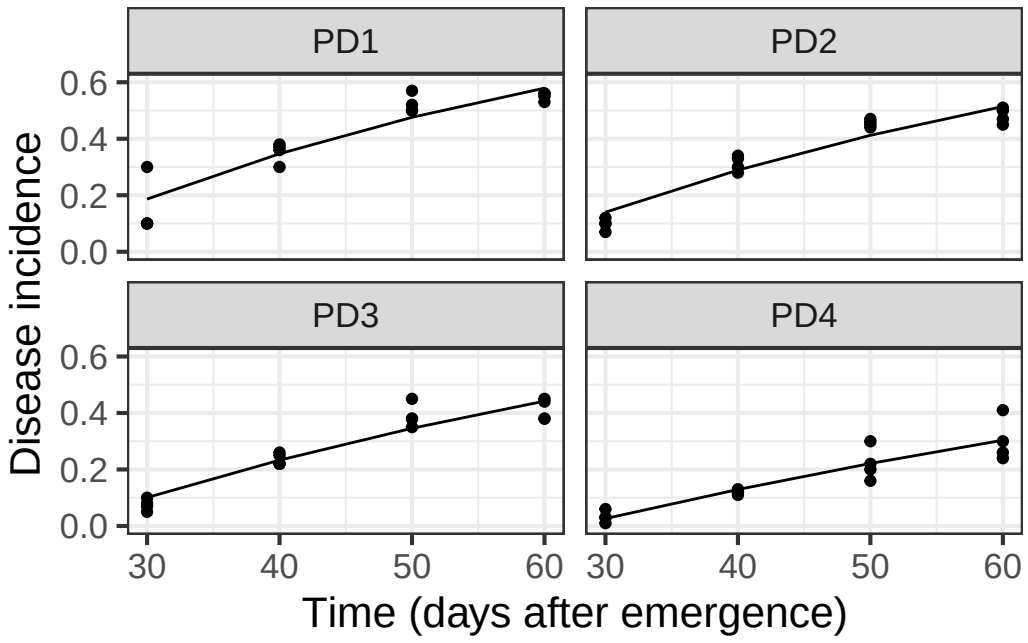
And now we extract values for each parameter estimated from the fit of the monomolecular model.

```
  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.

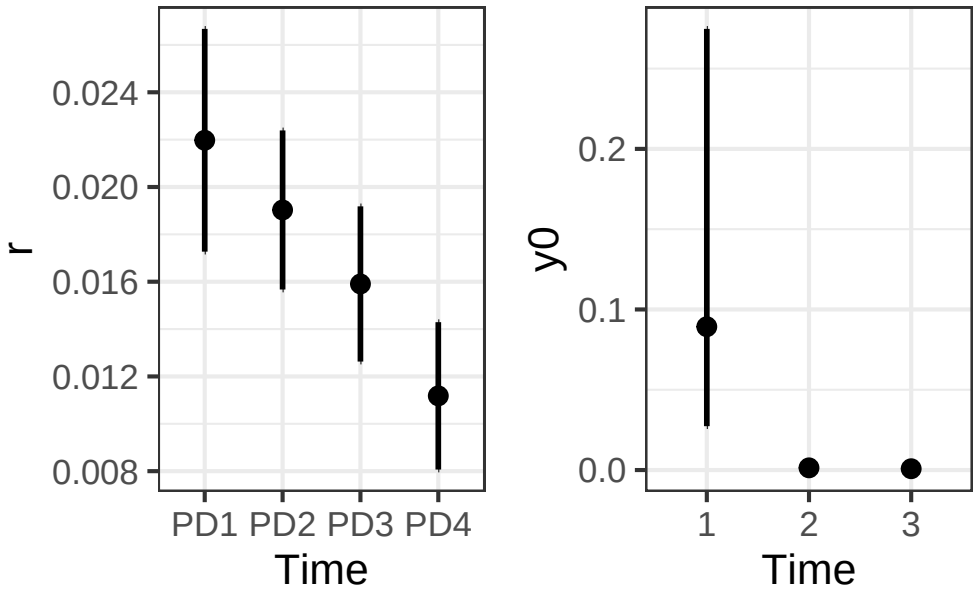
```
lin1$Data |>
  filter(model == "Monomolecular") |>
  ggplot(aes(time, predicted)) +
  geom_point(aes(time, y)) +
  geom_line(size = 0.5) +
  facet_wrap(~treat) +
  coord_cartesian(ylim = c(0, 0.6)) + # set the max to 0.6
  labs(
    y = "Disease incidence",
    x = "Time (days after emergence)"
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



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.

```
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
```



8.0.5.3.2 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`.

```
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)
```

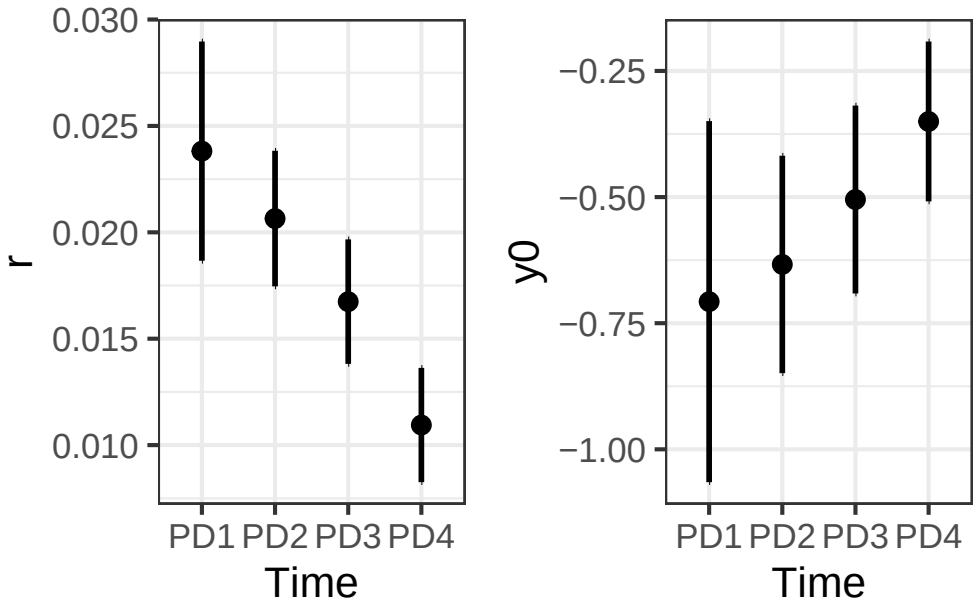
```
   treat         model       CCC        RSE best_model
1    PD1 Monomolecular 0.9382991 0.06133704          1
2    PD1      Gompertz 0.9172407 0.06986307          2
3    PD1      Logistic 0.8957351 0.07700720          3
4    PD1   Exponential 0.8544194 0.08799512          4
5    PD2 Monomolecular 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.

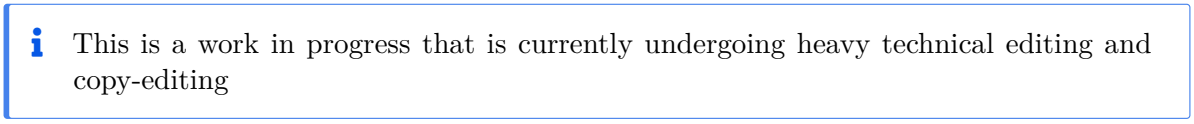
```
  nlin1$Parameters |>
    filter(model == "Monomolecular") |>
    select(treat, y0, r)
```

```
  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

9 Simulation



This is a work in progress that is currently undergoing heavy technical editing and copy-editing

```
library(tidyverse) # essential packages
library(cowplot) # for themes
library(deSolve)
theme_set(theme_bw(base_size = 16)) # set global theme
```

```
HLIR_fun <- function(t, y, par) {
  # Variables
  H <- y[1]
  L <- y[2]
  I <- y[3]
  R <- y[4]
  beta <- par$beta
  gama <- par$gama
  mu <- par$mu

  # Right hand side of the model
  dH <- -beta * H * I
  dL <- beta * H * I - gama * L
  dI <- gama * L - mu * I
  dR <- mu * I
  return(list(c(dH, dL, dI, dR)))
}
```

9.1 Setting up

```
# Set up parameters
beta <- 0.002 # Per capita rate of infection of susceptible hosts
gama <- 1 / 5 #  Rate at which exposed (i.e., latently infected) hosts become infectious
delta <- 15 # infectious period
mu <- 1 / delta
InitCond <- c(1000, 1, 0, 0)
```

9.2 Solving equation

```
steps <- seq(1, 60, by = 1)
parms <- list(beta = beta, gama = gama, mu = mu)
HLIR <- ode(InitCond, steps, HLIR_fun, parms)
epidemics <- data.frame(time = HLIR[, 1], H = HLIR[, 2], L = HLIR[, 3], I = HLIR[, 4], R =
```

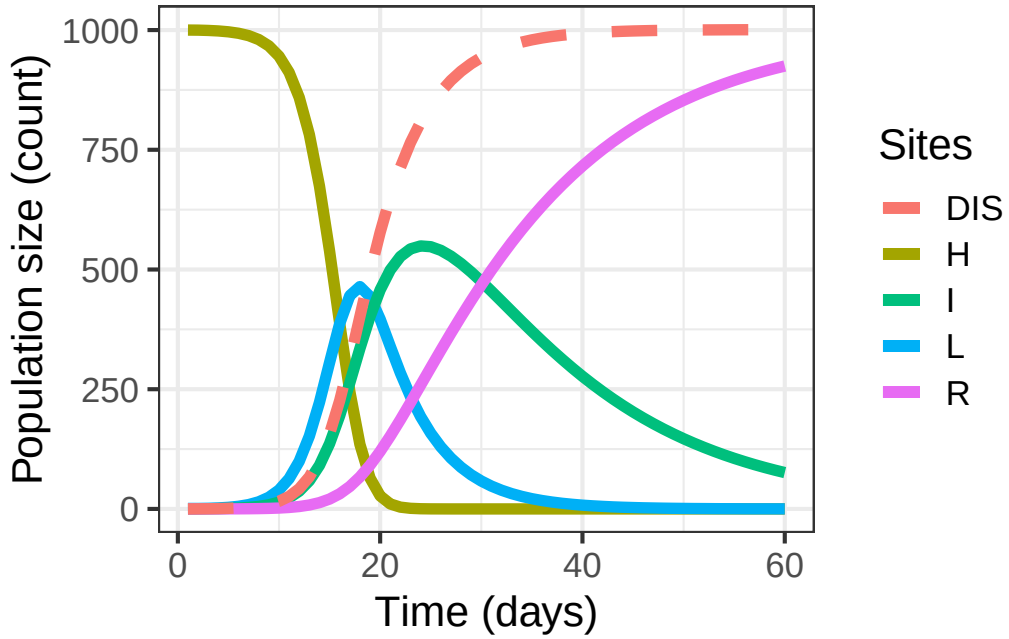
9.3 Visualizing HLIR output

```
library(ggthemes)
```

```
Attaching package: 'ggthemes'

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

    theme_map
```

```
p1 <- epidemics %>%
  ggplot() +
  geom_line(aes(time, H, color = "H"), size = 2) +
  geom_line(aes(time, L, color = "L"), size = 2) +
  geom_line(aes(time, I, color = "I"), size = 2) +
  geom_line(aes(time, R, color = "R"), size = 2) +
  geom_line(aes(time, I+R, color = "DIS"), size =2, linetype = 2)+
  labs(y = "Population size (count)", x = "Time (days)", color = "Sites")
```

```
p1
```


Population size (count)
1000
750
500
250
0
0
20
40
60
Time (days)
Sites
DIS
H
I
L
R

Part III

Spatial analysis

10 Spatial gradients

> ℹ This is a work in progress that is currently undergoing heavy technical editing and copy-editing

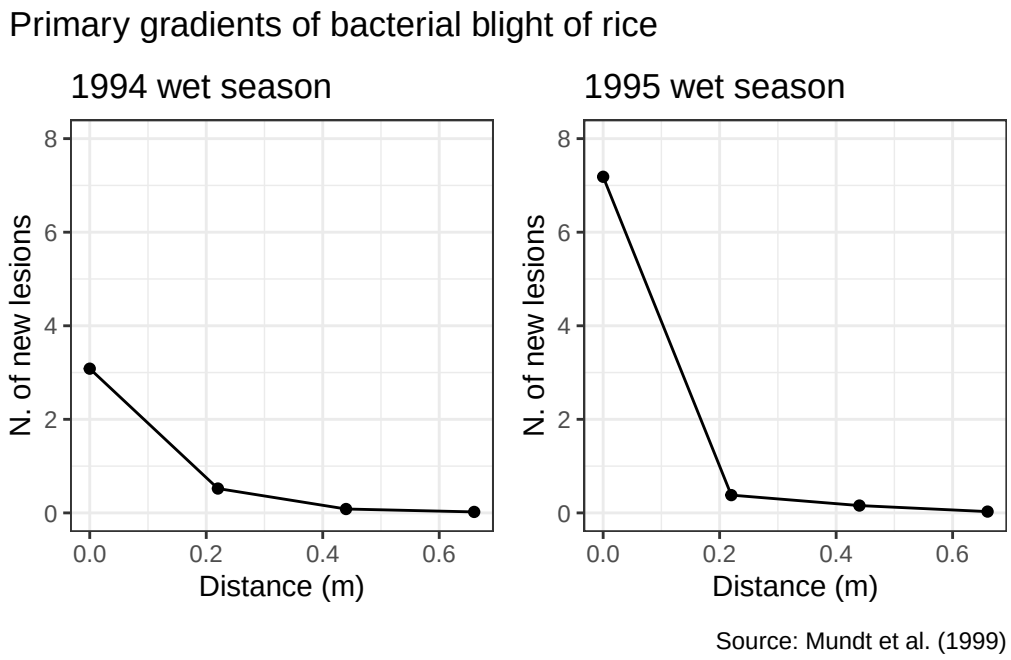
10.1 Introduction

The assessment of disease in space, in terms of changes in the intensity as it spreads over distance, is called **disease gradient**. In reality, it is the dispersal (migration) of the pathogen by various means (e.g. wind, vectors, rain, movement of infected material or human mediation) that promotes the spread of plant diseases within a field or across continents and generates the disease gradients. There are two kinds of gradients, the inoculum gradient where host availability is not necessarily required and the disease gradient where the three elements of the disease triangle are required. Let's see examples of actual disease gradients measured in the field with quite distinct patterns.

In the first example (Mundt et al. 1999), the objective of the authors was to measure the dispersal potential of the pathogenic bacteria *Xanthomonas oryzae* pv. *oryzae* that causes leaf blight on rice using experimental plots in the Philippines, during the wet seasons of 1994 and 1995. The data were made available in this tutorial. We enter the data manually and then produce two plots, one for each year.

```
library(tidyverse)
theme_set(theme_bw())
d <- c(0.0, 0.22, 0.44, 0.66) # distance in meters
y4 <- c(3.083, 0.521, 0.083, 0.021) # n. of new lesions in 1994 trial
y5 <- c(7.185, .380, 0.157, 0.028) # n. of new lesion in 1995 trial
xo <- data.frame(d, y4, y5)

g1 <- xo |>
  ggplot(aes(d, y4))+
  geom_point()+
  geom_line()+
  ylim(0,8)+
```

```
  labs(y = "N. of new lesions",
       x = "Distance (m)",
       title = "1994 wet season")

g2 <- xo |>
  ggplot(aes(d, y5))+
  geom_point()+
  geom_line()+
   ylim(0,8)+
  labs(y = "N. of new lesions",
       x = "Distance (m)",
       title = "1995 wet season")

library(patchwork)
(g1 | g2) +  plot_annotation(
  title = 'Primary gradients of bacterial blight of rice',
  caption = "Source: Mundt et al. (1999)")
```

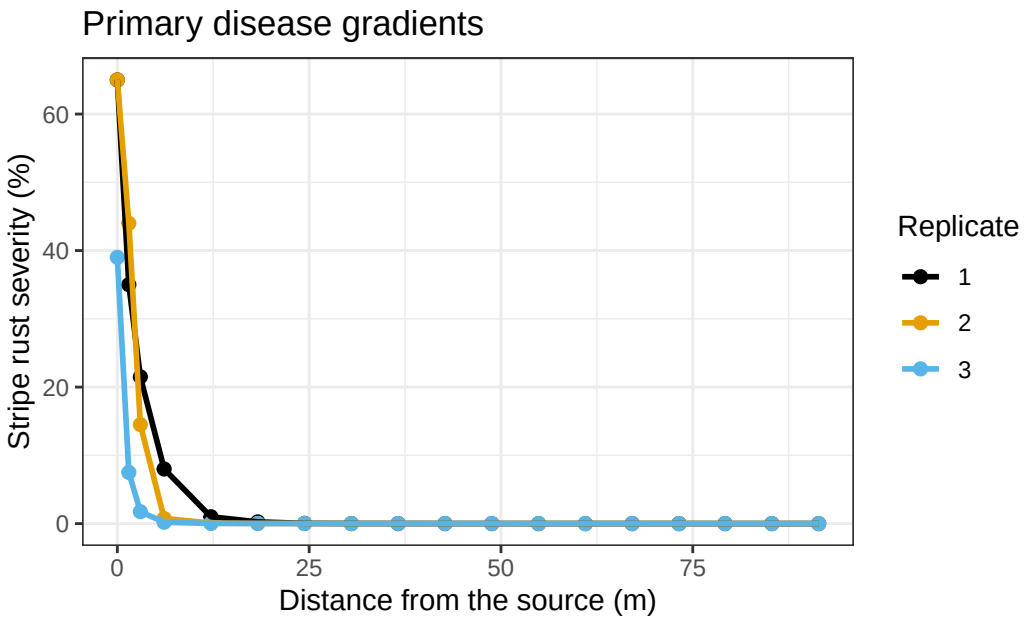


The second gradient is of stripe rust, caused by *Puccinia striiformis* f. sp. *tritici,* on wheat collected in a field experiment at Hermiston in 2002 (Sackett and Mundt 2005) and whose data was made freely available (Mikaberidze et al. 2015). Let's enter data manually as a tibble format. There are five columns. The first and second are distances in feet and meters,

respectively, from the source of infection. The other three columns are measures of stripe rust severity on replicated plots.

```
hermiston <-
  tibble::tribble(
  ~dist_f, ~dist_m,  ~`1`,  ~`2`,  ~`3`,
  0,       0,    65,    65,    39,
  5,     1.5,    35,    44,   7.5,
  10,      3,  21.5,  14.5,  1.75,
  20,    6.1,     8,  0.75,   0.2,
  40,   12.2,     1,  0.08, 0.025,
  60,   18.3,  0.25, 0.026, 0.015,
  80,   24.4, 0.035, 0.015, 0.009,
  100,   30.5,  0.01, 0.003, 0.008,
  120,   36.6, 0.008, 0.016,  0.01,
  140,   42.7, 0.003, 0.003,  0.01,
  160,   48.8, 0.001, 0.006, 0.006,
  180,   54.9, 0.001, 0.002, 0.002,
  200,     61, 0.001, 0.003, 0.004,
  220,   67.1, 0.001, 0.003, 0.002,
  240,   73.2, 0.001, 0.001,     0,
  260,   79.2, 0.001, 0.002,     0,
  280,   85.3, 0.001, 0.001,     0,
  300,   91.4, 0.001, 0.001, 0.001
  )
```

```
library(tidyverse)
library(ggthemes)
theme_set(theme_bw())
hermiston |>
  pivot_longer(3:5, names_to = "replicate", values_to = "severity") |>
  ggplot(aes(dist_m, severity, color = replicate))+
  geom_point(size = 2)+
  geom_line(size = 1)+
  scale_color_colorblind()+
  labs(title = "Primary disease gradients",
       x = "Distance from the source (m)",
       y = "Stripe rust severity (%)",
       color = "Replicate",
       caption = "source: Sackett et al. (2005)")
```



source: Sackett et al. (2005)

As it is clear in the above examples, in disease gradients, assuming that there is only a single source of inoculum, the intensity of the disease decreases more steeply within short distances of the source, and less steeply at greater distances until they reach zero or a low background level of occasional diseased plants.

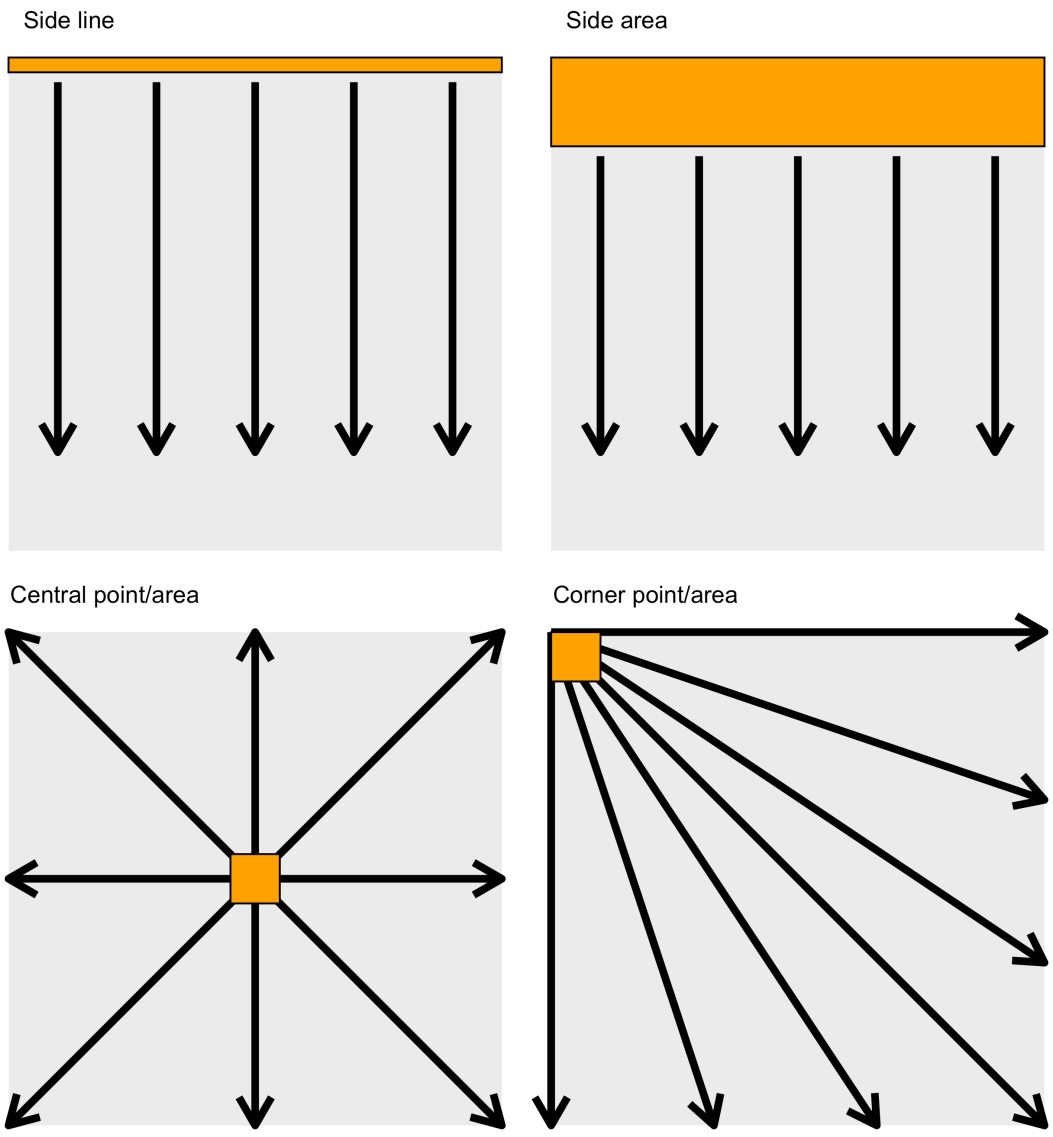
The shapes of the gradients are defined by mechanisms associated with the dispersal of the inoculum which depends on the biology of the pathogen but strongly to environmental factors that affect pathogen dispersal.

When studying disease gradients, researchers need to make sure that there is a well-defined single source of inoculum. In gradients, this is called a **focus** (where foci are deemed the plural), from where the inoculum originates. Three types of foci can be defined: point, line or area sources. While the point source can be a plant or group of plants at any position in the plot or field (center or corner), line and area sources are usually defined as one or more rows of diseased plants at one side of the plot or field.

```
library(ggplot2)
library(gganimate)
line <- ggplot(data.frame(c(1:10),c(1:10)))+
  annotate("rect", xmin = 0, xmax = 10, ymin = 0, ymax = 10, fill = "gray92")+
  annotate("rect", xmin = 0, xmax = 10, ymin = 9.7, ymax = 10, color = "black", fill = "or
  annotate("segment", size = 2, x = 1, xend = 1, y = 9.5, yend = 2, arrow = arrow())+
  annotate("segment", size = 2, x = 3, xend = 3, y = 9.5, yend = 2, arrow = arrow())+
```

```
  annotate("segment", size = 2, x = 5, xend = 5, y = 9.5, yend = 2, arrow = arrow())+
  annotate("segment", size = 2, x = 7, xend = 7, y = 9.5, yend = 2, arrow = arrow())+
  annotate("segment", size = 2, x = 9, xend = 9, y = 9.5, yend = 2, arrow = arrow())+
  ylim(0,10)+
  xlim(0,10)+
  coord_fixed()+
  theme_void()+
  labs(title = "      Side line")

area <- ggplot(data.frame(c(1:10),c(1:10)))+
  annotate("rect", xmin = 0, xmax = 10, ymin = 0, ymax = 10, fill = "gray92")+
  annotate("rect", xmin = 0, xmax = 10, ymin = 8.2, ymax = 10, color = "black", fill = "or
  annotate("segment", size = 2, x = 1, xend = 1, y = 8, yend = 2, arrow = arrow())+
  annotate("segment", size = 2, x = 3, xend = 3, y = 8, yend = 2, arrow = arrow())+
  annotate("segment", size = 2, x = 5, xend = 5, y = 8, yend = 2, arrow = arrow())+
  annotate("segment", size = 2, x = 7, xend = 7, y = 8, yend = 2, arrow = arrow())+
  annotate("segment", size = 2, x = 9, xend = 9, y = 8, yend = 2, arrow = arrow())+
  ylim(0,10)+
  xlim(0,10)+
  coord_fixed()+
  theme_void()+
  labs(title = "      Side area")

point_central <- ggplot(data.frame(c(1:10),c(1:10)))+
  annotate("rect", xmin = 0, xmax = 10, ymin = 0, ymax = 10, fill = "gray92")+
  annotate("segment", size = 2, x = 5, xend = 10, y = 5, yend = 10, arrow = arrow())+
  annotate("segment", size = 2, x = 5, xend = 10, y = 5, yend = 5, arrow = arrow())+
  annotate("segment", size = 2, x = 5, xend = 10, y = 5, yend = 0, arrow = arrow())+
  annotate("segment", size = 2, x = 5, xend = 0, y = 5, yend = 0, arrow = arrow())+
  annotate("segment", size = 2, x = 5, xend = 0, y = 5, yend = 5, arrow = arrow())+
  annotate("segment", size = 2, x = 5, xend = 0, y = 5, yend = 10, arrow = arrow())+
  annotate("segment", size = 2, x = 5, xend = 5, y = 5, yend = 10, arrow = arrow())+
  annotate("segment", size = 2, x = 5, xend = 5, y = 5, yend = 0, arrow = arrow())+
   annotate("rect", xmin = 5.5, xmax = 4.5, ymin = 4.5, ymax = 5.5, color = "black", fill
  ylim(0,10)+
  xlim(0,10)+
  coord_fixed()+
  theme_void()+
  labs(title = "    Central point/area")

point_corner <- ggplot(data.frame(c(1:10),c(1:10)))+
```

```
  annotate("rect", xmin = 0, xmax = 10, ymin = 0, ymax = 10, fill = "gray92")+
  annotate("segment", size = 2, x = 0, xend = 10, y = 10, yend = 0, arrow = arrow())+
  annotate("segment", size = 2, x = 0, xend = 6.6, y = 10, yend = 0, arrow = arrow())+
  annotate("segment", size = 2, x = 0, xend = 3.3, y = 10, yend = 0, arrow = arrow())+
  annotate("segment", size = 2, x = 0, xend = 0, y = 10, yend = 0, arrow = arrow())+
  annotate("segment", size = 2, x = 0, xend = 10, y = 10, yend = 3.3, arrow = arrow())+
  annotate("segment", size = 2, x = 0, xend = 10, y = 10, yend = 6.6, arrow = arrow())+
  annotate("segment", size = 2, x = 0, xend = 10, y = 10, yend = 10, arrow = arrow())+
  annotate("rect", xmin = 0, xmax = 1, ymin = 9, ymax = 10, color = "black", fill = "orang
  ylim(0,10)+
  xlim(0,10)+
  coord_fixed()+
  theme_void()+
  labs(title = "    Corner point/area")

library(patchwork)
p_gradients <- (line | area)/
(point_central | point_corner)

ggsave("imgs/gradients.png", width =9, height =9, bg = "white")
```

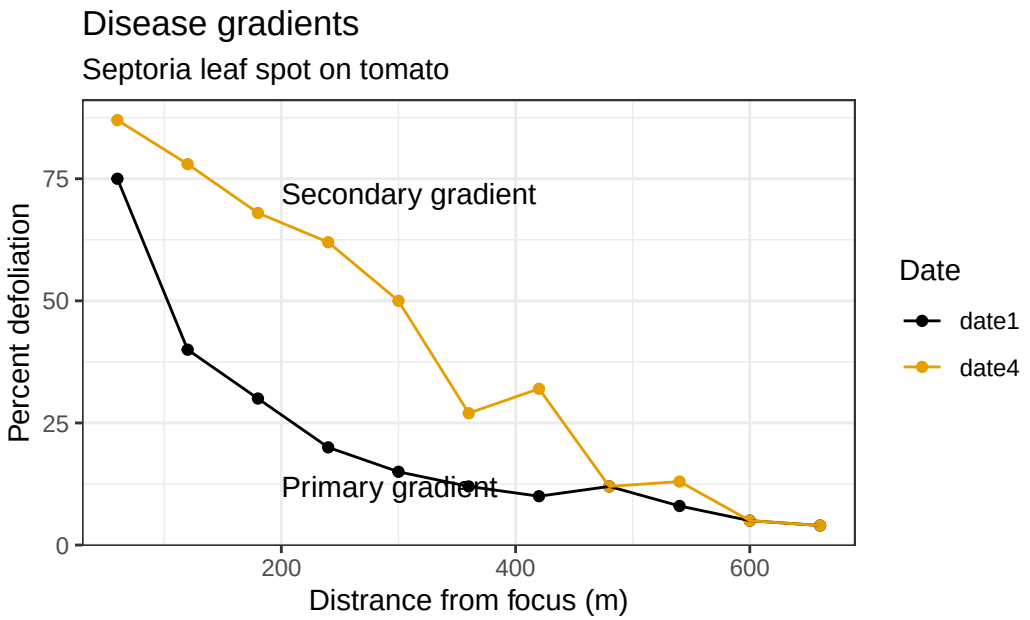


The resulting gradients can be classified in two types: primary or secondary. The primary gradient originates only from the initial focus, while the secondary gradient originates from the movement of inoculum produced at previously infected (due to primary gradients) plants to other plants at increasing distances from the source. It is expected that a mix of both kinds of gradients exists as the disease increases over time.

As an example of primary and secondary gradients, let's visualize the gradients of Septoria leaf spot, caused by *Septoria lycopersici*, on tomato (Parker et al. 1997). The gradients were measured during two times, thus enabling a comparison of primary and secondary dispersal/disease gradients. More details of the study and experimental approach were provided in

this tutorial. The data is entered below as a tribble and the plot produced using g*gplot2*.

```
septoria <-
tibble::tribble(
 ~d, ~date1, ~date4,
 60,    75,    87,
 120,    40,    78,
 180,    30,    68,
 240,    20,    62,
 300,    15,    50,
 360,    12,    27,
 420,    10,    32,
 480,    12,    12,
 540,     8,    13,
 600,     5,     5,
 660,     4,     4
             )

septoria |>
  pivot_longer(2:3, names_to = "date",
               values_to = "defoliation") |>
  ggplot(aes(d, defoliation, color = date))+
  geom_point()+
  geom_line()+
  scale_color_colorblind()+
  annotate(geom = "text", x = 200, y = 12,
           label = "Primary gradient", hjust = "left")+
   annotate(geom = "text", x = 200, y = 72,
           label = "Secondary gradient", hjust = "left")+
  labs(title = "Disease gradients",
       subtitle = "Septoria leaf spot on tomato",
       x = "Distrance from focus (m)",
       y = "Percent defoliation",
       color = "Date",
       caption = "Parker et al. (1997)")
```



Parker et al. (1997)

11 Gradient models

Similar to disease progress curves, models can be fitted empirically to observed disease gradient curves and provide insights into the mechanisms of inoculum dispersal and deposition, the source of inoculum, and the physical processes underlying dispersal.

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.

11.0.1 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.

```
library(tidyverse)
library(ggthemes)
library(patchwork)
library(cowplot) # for themes
```

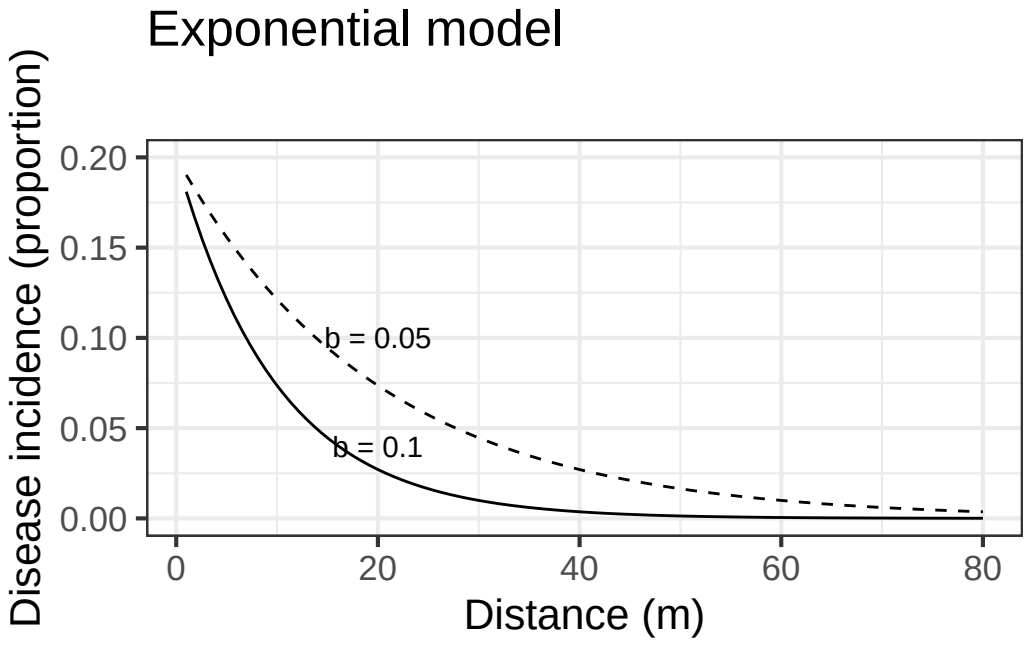
```
theme_set(theme_bw(base_size = 16)) # set global theme
```

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

```
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.

```
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)"
  )
```



11.0.2 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.

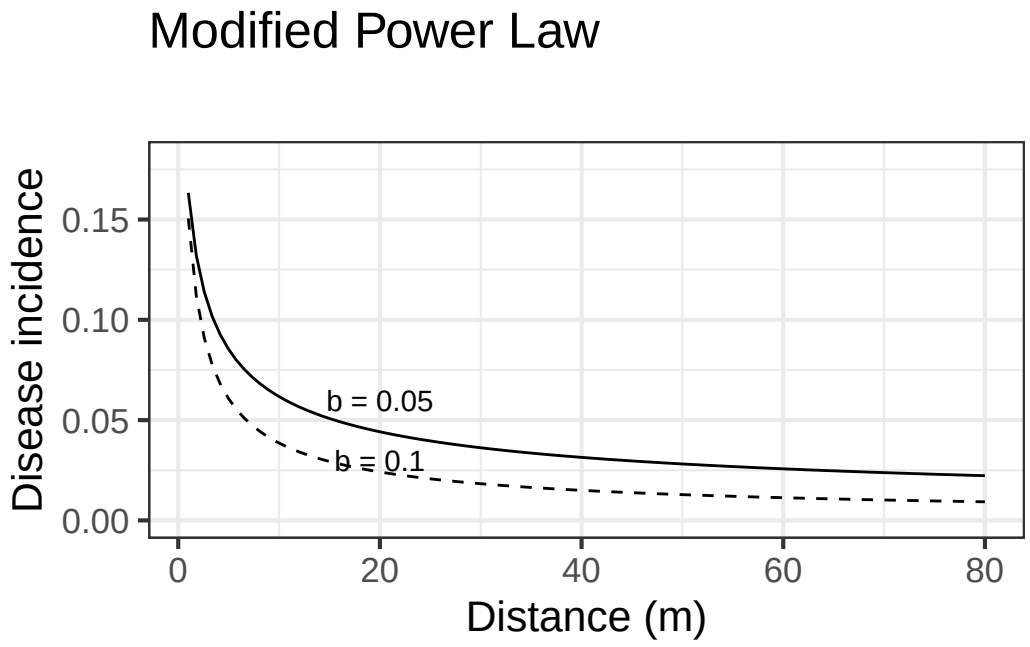
Once y at the distance zero from the source is undefined when using the power law model, this is usually modified by the addition of a positive constant C in x:

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

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.

11.1 Linearization of the models

11.1.1 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.

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

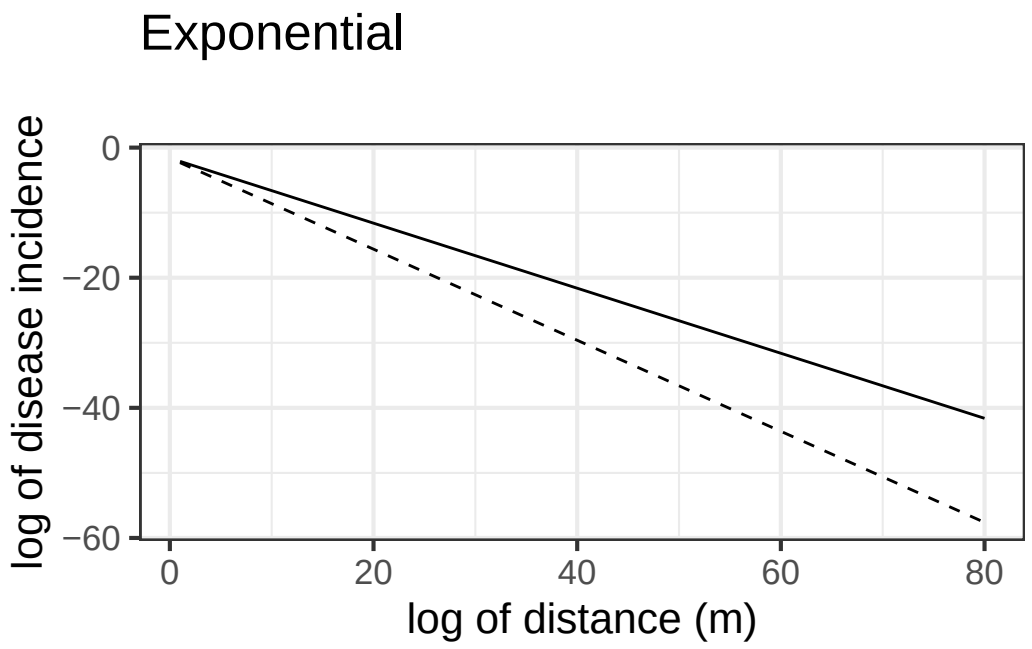
$ln(y) = ln(a_P) - b_E.ln(x + C)$

11.1.2 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 $ln(y)$.

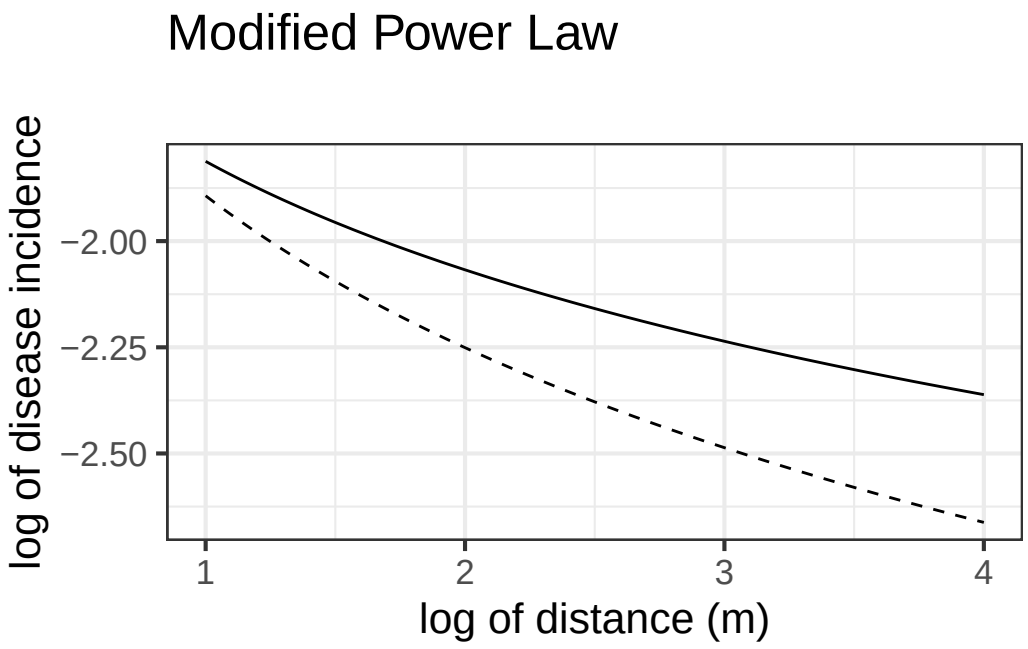
Follows the linearization of the modified power law model.

```
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"
  )
```


Modified Power Law
log of disease incidence
−2.00
−2.25
−2.50
1
2
3
4
log of distance (m)

12 Fitting gradient models

> This is a work in progress that is currently undergoing heavy technical editing and copy-editing

```
library(tidyverse)
library(ggthemes)
library(patchwork)
library(cowplot) # for themes
theme_set(theme_bw(base_size = 16)) # set global theme
```

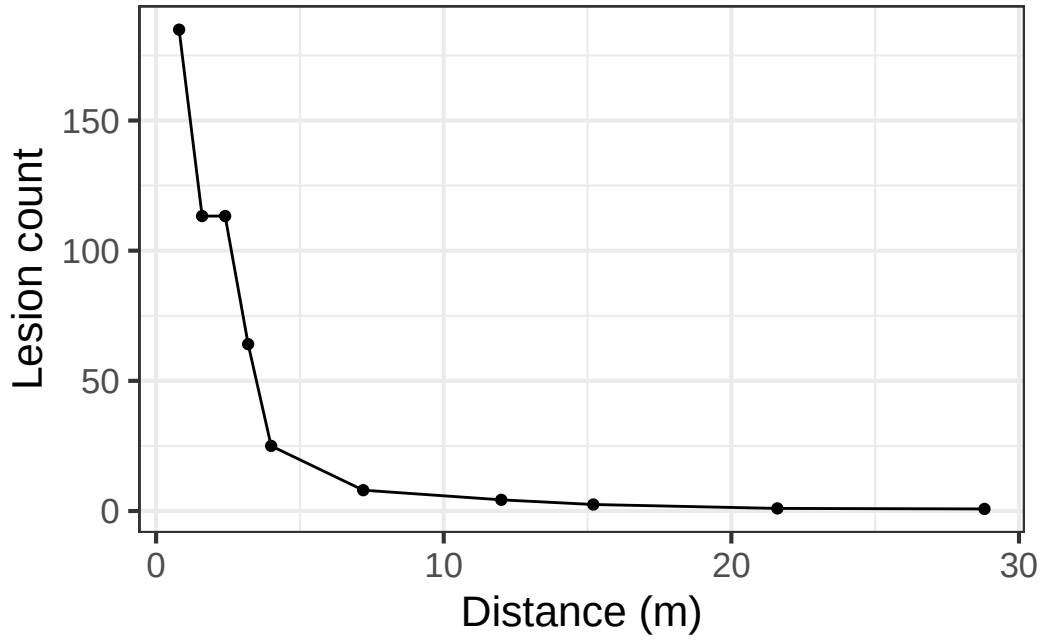
The hypothetical data below shows a gradient for the number of lesions counted at varying distances in meters from the source. Let's create two vectors, one for the distances x and the other for the lesion count Y, and then a dataframe by combining the two vectors.

```
# 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
```

12.1 Visualize the gradient

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



13 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 $C = 0$.

```
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 $C = 0.4$.

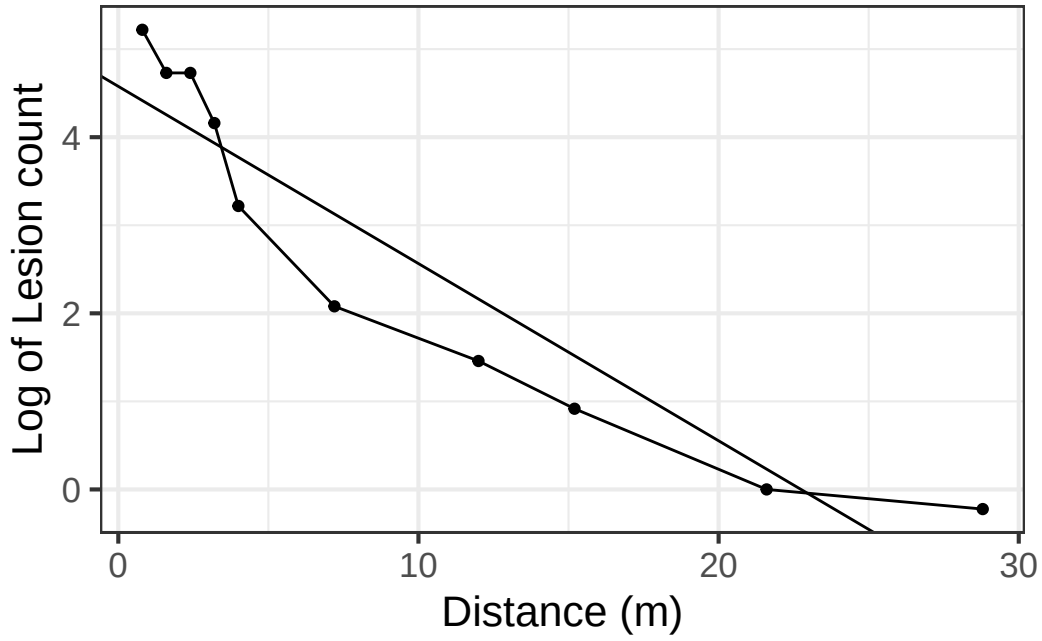
```r
reg_pm <- lm(log(Y) ~ log(x + 0.4), data = grad1)
summary(reg_pm)
```

```
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

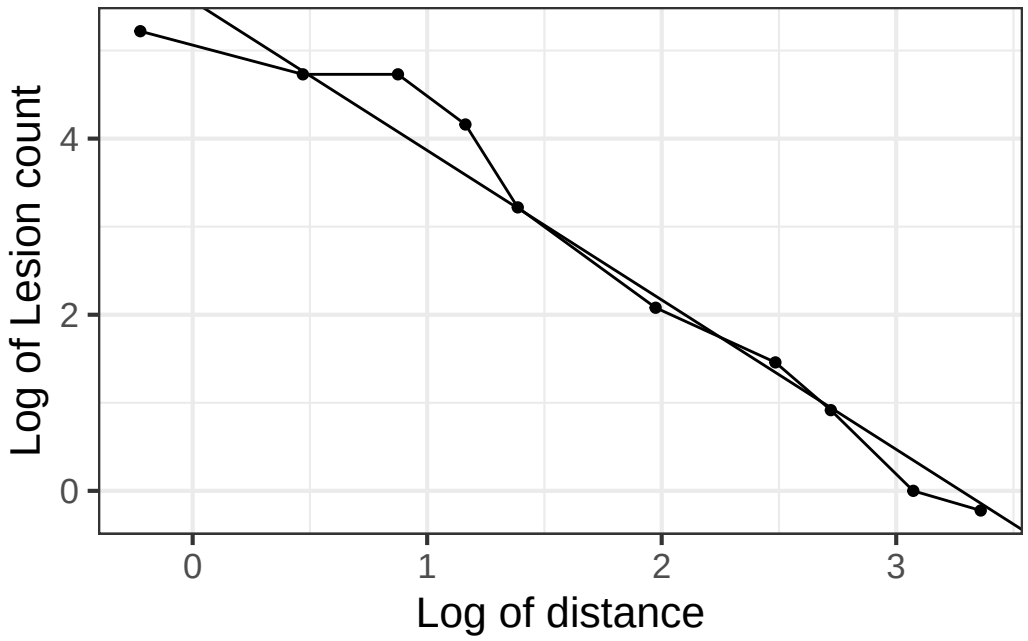
Exponential

```
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)")
```



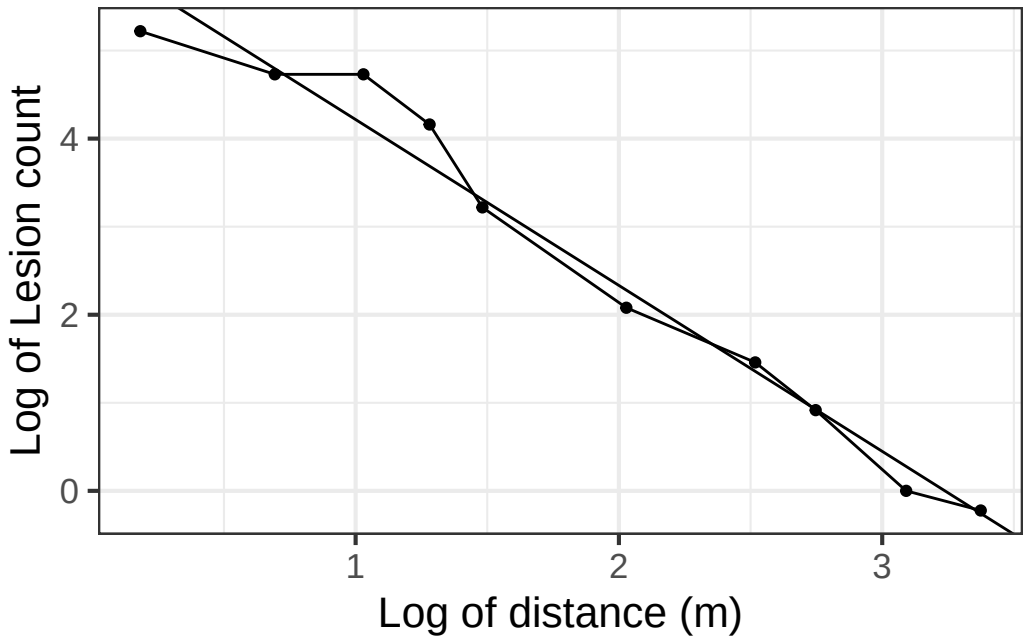
Power law model

```
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.

14 Spatial patterns

This is a work in progress that is currently undergoing heavy technical editing and copy-editing

14.1 Definitions

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. 2017a). 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.

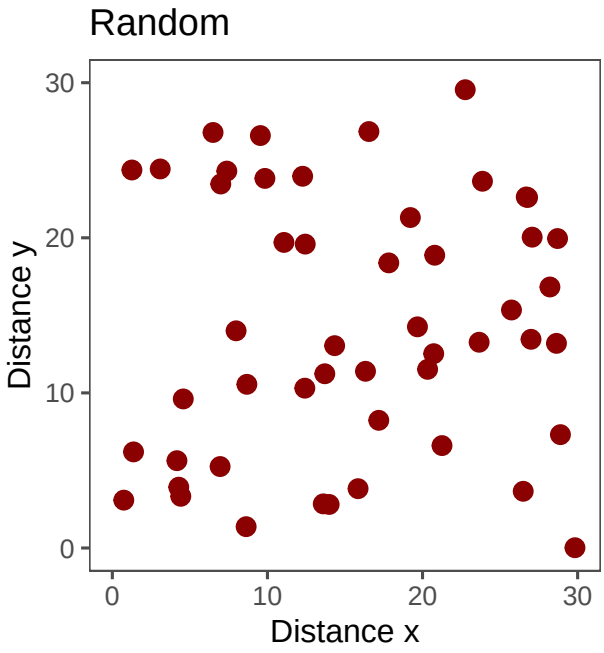
Let's simulate in R two vectors (x,y) for the positions of diseased unitss that follow a random or an aggregated pattern. For the random pattern, we use `runif`, a function which generates random deviates from the uniform distribution.

```
set.seed(123)          # for reproducibility
x <- runif(50, 0, 30)  # x vector
y <- runif(50, 0, 30)  # y vector
```

```r
dat <- data.frame(x,y) # dataframe for plotting
```

Now, the plot to visualize the random pattern.

```r
library(tidyverse)
library(ggthemes)
theme_set(theme_few())

pr <- dat |> # R base pipe operator
  ggplot(aes(x, y))+
  geom_point(size =3,
             color = "darkred")+
  ylim(0,30)+
  xlim(0,30)+
  coord_fixed()+
  labs(x = "Distance x", y = "Distance y",
       title = "Random")
pr
```



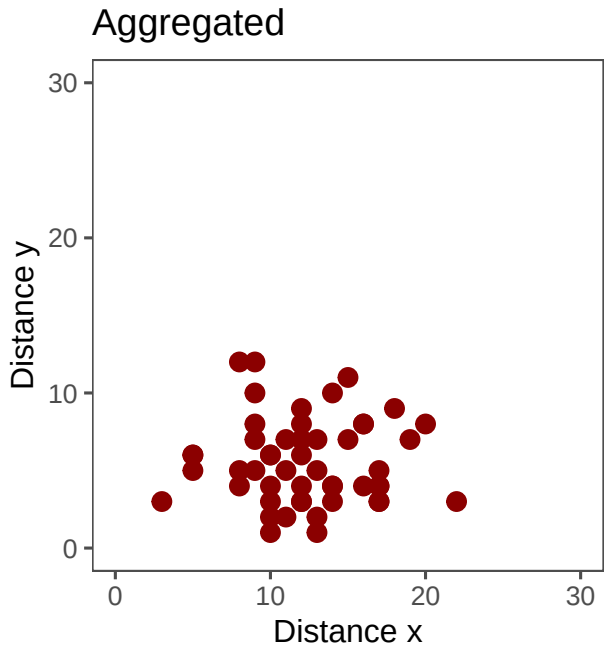
Now, we can generate new x and y vectors using `rnbinom` function which allows generating values for the negative binomial distribution (which should give rise to aggregated patterns) with parameters `size` and `prob`. Let's simulate 50 values with mean 12 and size 20 as dispersal

parameter.

```
x <- rnbinom(n = 50, mu = 12, size = 20)
y <- rnbinom(n = 50, mu = 5, size = 20)
dat2 <- data.frame(x, y)
```

This should give us an aggregated pattern.

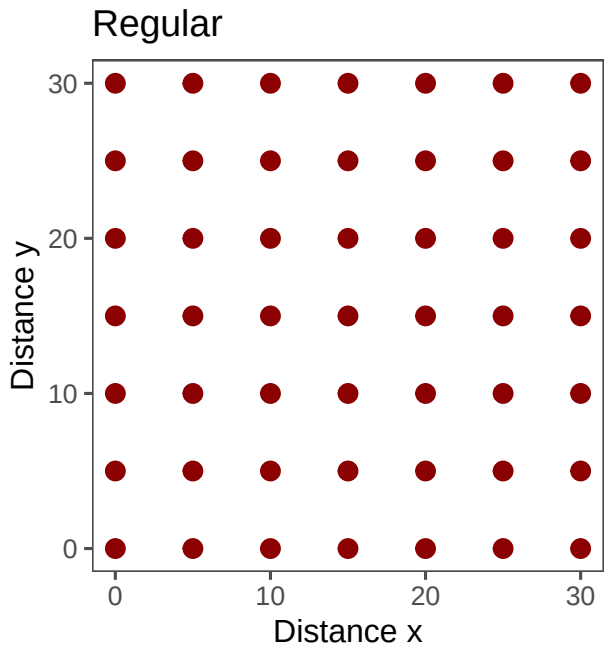
```
pag <- dat2 |>
  ggplot(aes(x, y))+
  geom_point(size = 3, color = "darkred")+
  ylim(0,30)+
  xlim(0,30)+
  coord_fixed()+
  labs(x = "Distance x", y = "Distance y",
       title = "Aggregated")
pag
```



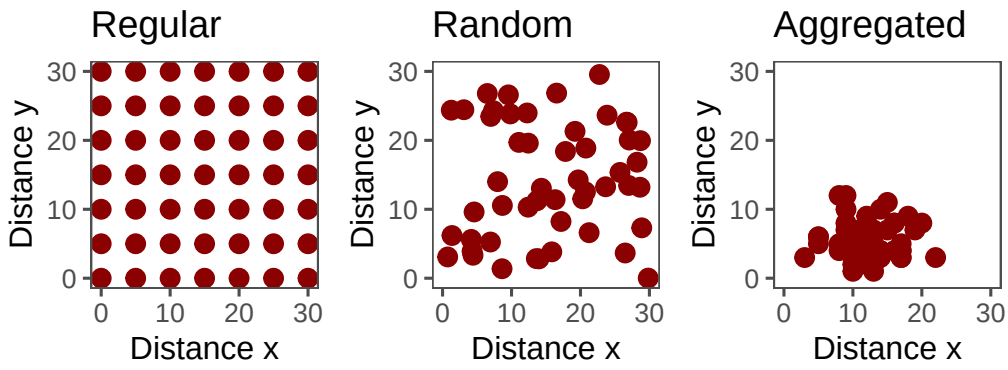
A rare pattern found in nature is the regular pattern, but it may be generated artificially by the man when conducting experimentation. Follows a code to produce the regular pattern.

```
x <- rep(c(0,5,10,15,20, 25, 30, 35, 40, 45), 5)
y <- rep(c(0, 5, 10, 15, 20, 25, 30, 35, 40, 45), each = 10)
dat3 <- data.frame(x, y)

preg <- dat3 |>
  ggplot(aes(x, y))+
  geom_point(size = 3, color = "darkred")+
  ylim(0,30)+
  xlim(0,30)+
  coord_fixed()+
  labs(x = "Distance x", y = "Distance y",
       title = "Regular")
preg
```

```
Warning: Removed 51 rows containing missing values (geom_point).
```




```
library(patchwork)

preg + pr + pag
```




```
ggsave("imgs/spatial.png", width = 10, height = 4)
```

We can create an animation showing the three patterns using *gganimate* package

```
rand <- dat |>
  mutate(i = "random")

agg <- dat2 |>
  mutate(i = "aggregated")

reg <- dat3 |>
  mutate(i = "regular")

patterns <- rbind(agg, rand, reg)

library(gganimate)

patterns |>
ggplot(aes(x, y, color = factor(i)))+
  geom_point(size = 4, color = "darkred")+
  coord_fixed()+
```