R4PDE

Emerson M. Del Ponte

Table of contents

Welcome

R for Plant Disease Epidemiology (R4PDE) is a dynamic book project rooted in the teachings of the annual graduate course, FIP 602 - Plant Disease Epidemiology, a key part of the curriculum in the Graduate Program in Plant Pathology at Universidade Federal de Viçosa.

Designed for those passionate about studying and modeling plant disease epidemics with R, the book offers an exploration of diverse methods for describing, visualizing, and analyzing epidemic data collected over time or space. Readers should ideally have a foundational knowledge of R to best utilize the examples.

However, R4PDE is not a resource for learning data science through R, as there are already well-established books such as R for data science for that purpose. Portuguese-speaking readers are recommended Análises Ecológicas no R and Software R para avaliação de dados experimentais as excellent R learning resources, with an added focus on statistics using R.

The book often draws upon data and replicates analyses from *The Study of Plant Disease Epidemics* (Madden et al. 2007d). A mix of general and specific R packages are utilized to conduct common plant disease epidemiology data analysis, notably {epifitter} and {epiphy}, both designed by plant pathologists. In conjunction with this book, a new R package {r4pde} has been developed and can be installed from GitHub using:

```
#install.packages("remotes")

remotes::install_github("emdelponte/r4pde")
```

As a work in progress, the book is frequently updated and edited. The website is free to use, licensed under a Creative Commons licence, and the code for all analyses can be found on GitHub. While there are no immediate plans for a printed version, it is under consideration as the book further develops. The website is hosted by https://www.netlify.com/.

Contributions to R4PDE are subject to a Contributor Code of Conduct, and by contributing, you agree to adhere to its terms.

Acknowledgements

To Helen Pennington for allowing me to use her painting of coffee leaf rust as book cover. Also to those who have contributed fixes and improvements via pull request or other form of contact: Adam Sparks (@adamhsparks), Remco Stam (@remco-stam), Tiago Olivoto (@TiagoOlivoto), and Monalisa De Cól (@Monalisacdc)

About the author

Emerson M. Del Ponte is a Professor at the Departamento de Fitopatologia, Universidade Federal de Viçosa in Brazil. His academic journey includes a DSc in Plant Pathology, obtained from Universidade Federal de Pelotas in 2004, and a year-long visit to Cornell University in the Bergstrom Lab. Following this, he spent nearly two years as a postdoctoral associate working on a project related to disease risk assessment and prediction at the Yang Lab, Iowa State University. This experience led him to the Universidade Federal do Rio Grande do Sul, Brazil, where he joined as an assistant professor of plant pathology.

Emerson Del Ponte is a fervent advocate for an open and reproducible research model and culture, which he believes can lead to more accessible, transparent, and reliable scientific knowledge. This belief inspired him co-founding the Open Plant Pathology initiative alongside Adam Sparks. In his Lab, students use the R language for all statistics and data science-related activities. All data and computational codes generated during the research are made accessible before the peer-review process. The code can be located on GitHub.

1 Introduction

1.1 Defining plant disease

Disease in plants can be defined as *any malfunctioning of host cells and tissues that results from continuous irritation by a pathogenic agent or environmental factor and leads to development of symptoms* (Agrios 2005a). When caused by pathogenic agent, the disease results from the combination of three elements: susceptible host plant, a virulent pathogen, and favorable environmental conditions - the famous disease triangle. 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. 2007d).

There exist numerous iterations of the disease triangle, incorporating additional elements (e.g., human intervention and time) as points and/or dimensions to provide a more comprehensive representation of an epidemic (Agrios 2005b). We find the disease prism particularly illustrative, where a sequence of stacked triangles represent the evolution of a plant disease through time (Francl 2001).

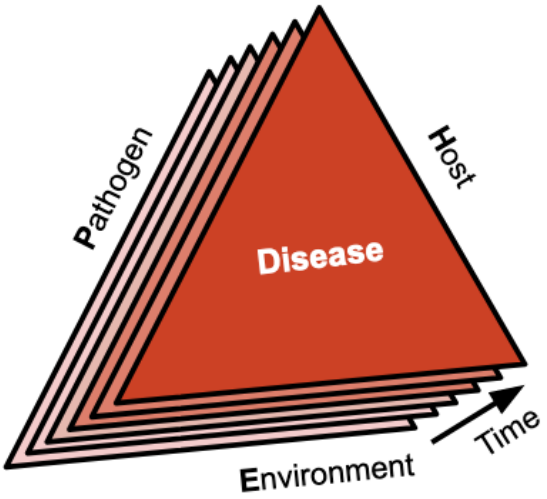


Figure 1.1: The plant disease prism as a model of plant disease epidemics

1.2 Importance of epidemics

Epidemics bear significant economic importance due to their potential to decrease crop yields, diminish product quality, and escalate control costs, contingent on their intensity level. Numerous historical examples of widespread epidemics, reaching pandemic levels and resulting in catastrophic effects on crops, have been documented (Agrios 2005b). The Irish potato famine of 1845–1847, caused by the late blight pathogen (*Phytophthora infestans*), is a famous example of a well-documented pandemic. This disease notably altered the course of history in Europe and the United States, and was pivotal in the evolution of the science of plant pathology. During the 1840s, the pathogen ravaged potato crops, which were a dietary staple for the Irish. The disease outbreak was triggered by the introduction of a novel, virulent pathogen population that found suitable environmental conditions (cool and wet weather) for infection and development within a dense population of susceptible hosts.

However, there are several reasons why devastating epidemics may continue to unfold. Recent history has seen severe epidemics reaching pandemic levels due to the incursion of pathogens into regions where they had previously been absent (refer to Box 1). Alternatively, new pathogenic strains might emerge as a result of factors driving genetic diversity within the local pathogen population. A case in point is the Ug99 strain of the wheat stem rust, which poses a significant threat to global wheat production. First identified in Uganda in 1998, an asexual lineage has propagated through Africa and the Middle East, causing catastrophic epidemics. Research suggests that Ug99 emerged via somatic hybridization and nuclear exchange between isolates from different lineages (Li et al. 2019). Finally, disease emergence or re-emergence can be influenced by shifts in climatic patterns. For instance, the Fusarium head blight of wheat caused by the fungus *Fusarium graminearum*. In Southern Brazil, the increased frequency of severe epidemics resulting in greater yield loss since the early 1990s has been linked to alterations in rainfall patterns across decades (Duffeck et al. 2020).

i Box 1: Diseases on the move

In Brazil, the soybean rust pathogen (*Phakopsora pachyrhizi*) first reached southern Brazil in 2002 (Yorinori et al. 2005). The disease spread to all production regions of the country in the following few years, severely reducing yields. To overcome the problem, farmers have relied on massive applications of fungicides on soybeans, which dramatically increased the production costs with the need for sequential fungicide sprays to combat the disease. Total economic loss have been estimated at around US$ 2 billion yearly (Godoy et al. 2016). More recently, wheat blast, a disease that originated in the south of Brazil in 1984, and have been restricted to South America, was firstly spotted in South Asia, Bangladesh, in 2016. Blast epidemics in that occasion devastated more than 15,000 ha of wheat and reduced yield of wheat in the affected field up to 100% (Malaker et al. 2016; Islam et al. 2019). The disease was later found in Zambia, thus also becoming a threat to wheat production in Africa (Tembo et al. 2020). In Brazil, the wheat blast disease is

a current threat to expansion of wheat cultivation in the tropics(Cruz and Valent 2017).

1.3 History of Epidemiology

Botanical epidemiology, or the study of plant disease epidemics, is a discipline with roots tracing back to the early 1960s. However, its origins can be linked to events from centuries and decades prior. For instance, in 1728, Duhamel de Monceau presented the earliest known epidemiological work on a disease, referred to as 'Death,' that afflicted saffron crocus (*Rhizoctonia violacea*). Fast forward to 1858, a textbook detailing plant diseases, written by Julius Kuhn, made its debut, introducing the concept of an epidemic as illustrated by the Irish late blight epidemics of 1845-46. Subsequently, in 1901, H.M. Ward adopted an ecological perspective to the study of plant diseases in his seminal book, Disease in Plants. By 1946, Gäumann penned the first book exclusively devoted to plant disease epidemiology.

Further evolution of this field was marked by the publication of a chapter titled "Analysis of Epidemics" by J.E. Vanderplank in Plant Pathology, vol. 3, edited by Horsfall and Dimond, in 1960. Vanderplank elaborated on his pioneering ideas in his 1963 book, "Plant Diseases: Epidemics and Control"(Vanderplank 1963). He is universally recognized as the foundational figure of plant disease epidemiology (Zadoks and Schein 1988; Thresh 1998), his landmark book being the first to comprehensively describe and quantify plant disease epidemics, and offering a theoretical framework for epidemic analysis.

In the same year, the first International Epidemiology Workshop was convened in Pau, France. This event constitutes an important milestone in the historical narrative, significantly contributing to the molding of this emergent discipline.

The **International Epidemiology Workshop (IEW)** is the principal working group of plant disease epidemiology. This is an organization with a rich history whose members have met approximately every 5 years since 1963. Thus far, 13 meetings have been organized/planned:

1963 - Pau, France
1971 - Wageningen, The Netherlands
1979 - Penn State, United States
1983 - NC State, Raleigh, United States
1986 - Jerusalem, Israel
1990 - Giessen, Germany
1994 - Papendal, The Netherlands
2001 - Ouro Preto, Brazil
2005 - Landerneau, France
2009 - Cornell, Geneva, United States
2013 - Beijing, China

Figure 1.2: Group photo of the First International Epidemiology Workshop

2018 - Lillehammer, Norway
2024 - Iguassu Falls, Brazil

1.4 Other resources

1.4.1 Books

2006 - The Epidemiology of Plant Diseases
2007 - The Study of Plant Disease Epidemics
2017 - Exercises in Plant Disease Epidemiology
2017 - Application of Information Theory to Epidemiology
2020 - Emerging Plant Diseases and Global Security

1.4.2 Online tutorials

Ecology and Epidemiology in R
Plant Disease Epidemiology - Temporal aspects
Simulation Modeling in Plant Disease Epidemiology and Crop Loss Analysis

1.4.3 Software

Epicrop - Simulation Modeling of Crop Diseases using a SEIR model

Part I

Epidemic data

2 Disease variables

2.1 Disease quantification

Studies on the temporal progression or spatial spread of epidemics cannot be conducted without field-collected data, or, in some cases, simulated data. The study of plant disease quantification, termed Phytopathometry, is a subdivision of plant pathology concerned with the science of disease measurement. It has strong ties to the field of epidemiology (Bock et al. 2021b).

Traditionally, disease quantification has been executed through visual evaluation. However, the past few decades have witnessed significant advancements in imaging and remote sensing technologies (which don't necessitate contact with the object), leaving a profound impact on this field. As such, disease quantity can now be gauged through estimation (visually, by the human eye) or measurement (through remote sensing technologies such as RGB, MSI, and HSI) Figure 2.1.

While the utilization of digital or remote sensing technology for disease measurement or estimation provides a more objective approach, visual assessment is largely subjective. It is known to vary among human raters, as these raters differ in their innate abilities, training, and how they are influenced by the chosen method (e.g., scales). Disease is estimated or measured on a specimen within a population, or on a sample of specimens drawn from that population. The specimen in question can be a plant organ, an individual plant, a group of plants, a field, or a farm. The specific specimen type also determines the terminology used to describe disease quantity.

inally, while developing new or refining existing disease assessment methods, it is crucial to evaluate the reliability of the assessments made by different raters or instruments, as well as their accuracy—specifically, how close the estimations or measurements are to the reference (or gold standard) values. Several methods are available for assessing the reliability, precision, and accuracy of these estimates or measurements (see definitions). The choice of methods depends on the objective of the work, but largely on the type or nature of the data. These considerations will be further discussed.

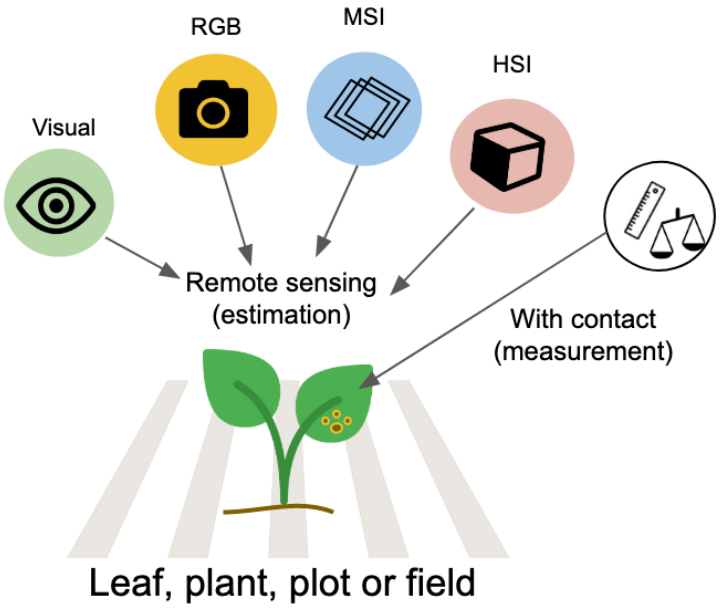


Figure 2.1: Different approaches used to obtain estimates or measures of plant disease. RGB = red, green, blue; MSI = multispectral imaging; HSI = hyperspectral imaging.

2.2 Disease variables

A common term used to reference the quantity of disease, irrespective of how it is expressed, is 'disease intensity'. This term, however, has minimal practical value as it only implies that the disease is more or less "intense". We require more specific terminology to standardize the reference to disease quantity and methodology. One of the primary tasks in disease assessment is classifying each specimen, often in a sample or within a population, as diseased or not diseased. This binary (yes/no or 1/0) evaluation may sufficiently express disease intensity if the goal is to ascertain the number or proportion of diseased specimens in a sample or a population.

This discussion brings us to two terms: disease incidence and prevalence. Incidence is typically used to denote the proportion or number (count) of plants (or their organs) deemed as observational units at the field scale or below. On the other hand, prevalence refers to the proportion or number of fields or farms with diseased plants within a larger production area or region (Nutter et al. 2006) Figure 2.2. Therefore, prevalence is analogous to incidence, with the only difference being the spatial scale of the sampling unit.

In many instances, it's necessary to determine the degree to which a specimen is diseased, a concept defined as disease severity. In certain contexts, severity is narrowly defined as the proportion of the unit that exhibits symptoms (Nutter et al. 2006). However, a more expansive

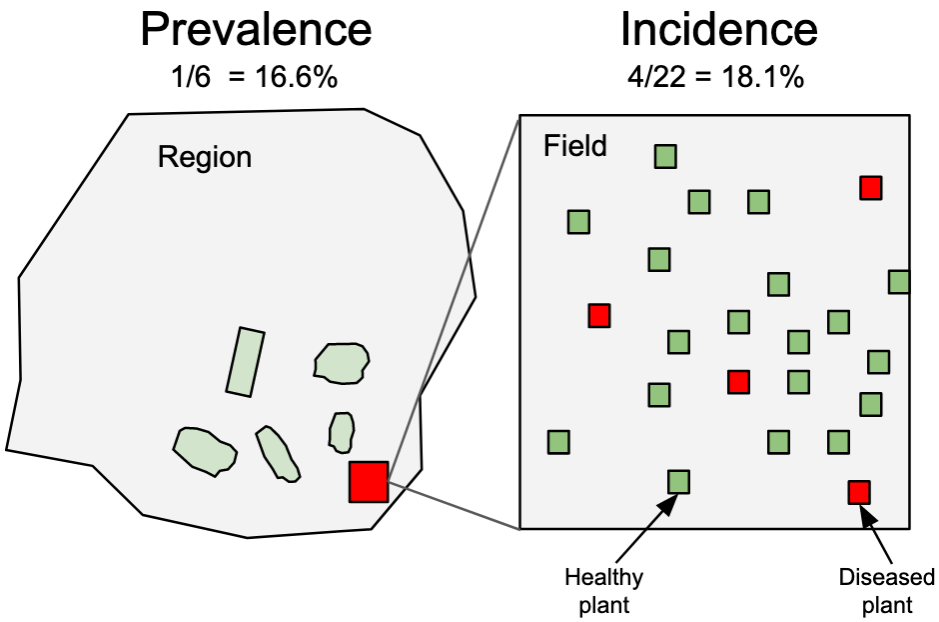


Figure 2.2: Schematic representation of how prevalence and incidence of plant diseases are calculated depending on the spatial scale of the assessment

view of severity includes additional metrics such as nominal or ordinal scores, lesion count, and percent area affected (ratio scale). Ordinal scales are broken down into rank-ordered classes (see specific section), defined based on either a percentage scale or descriptions of symptoms (Bock et al. 2021b). Occasionally, disease is expressed in terms of (average) lesion size or area, which could be regarded as a measure of severity. These variables represent different levels of measurements that provide varying degrees of information about the disease quantity - from low (nominal scale) to high (ratio scale) Figure 2.3.

2.3 Data types

The data used to express disease as incidence or any form of severity measurements can be discrete or continuous in nature.

Discrete variables are countable (involving integers) at a particular point in time. In other words, only a finite number of values (nominal or ordinal) is possible, and these cannot be subdivided. For instance, a plant or plant part can be either diseased or not diseased (nominal data). It's not possible to count 1.5 diseased plants. Furthermore, a plant classified as diseased may exhibit a certain number of lesions (count data), or be categorized into a specific severity class (ordinal data, common in ordinal scales, e.g., 1-9). Disease data in the form of counts

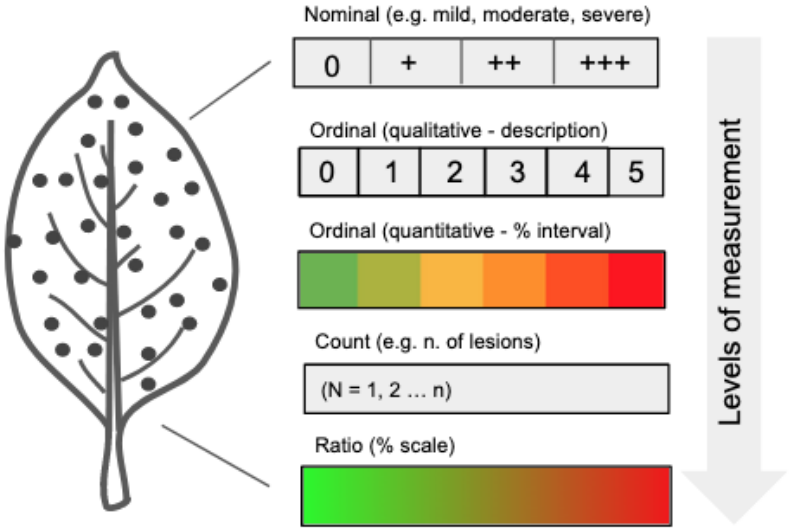


Figure 2.3: Scales and associated levels of measurement used to describe severity of plant diseases

often relates to the number of infections per sampling units. Most commonly, these counts refer to the assessed pathogen population, such as the number of airborne or soilborne propagules.

In contrast to discrete variables, continuous variables can be measured on a scale and can assume any numeric value between two points. 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 on a percent scale (% diseased area) can take any value between non-zero and 100%. Although incidence at the individual level is discrete, at the sample level it can be treated as continuous, as it can assume any value in proportion or percentage.

Disease variables can also be characterized by a statistical distribution, which are models that provide the probability of a specific value (or a range of values) being drawn from a particular distribution. Understanding statistical or mathematical distributions is a crucial step in improving our grasp of data collection methods, experiment design, and data analysis processes such as data summarization or hypothesis testing.

2.4 Statistical distributions and simulation

2.4.1 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)
library(r4pde)


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 = "#339966",
                 bins = 10) +
  facet_wrap( ~ P) +
  theme_r4pde()
```

2.4.2 Beta distribution

In many studies, it's often useful to express these quantities as a proportion of the total population or sample size, rather than absolute numbers. This helps standardize the data, making it easier to compare between different populations or different time periods.

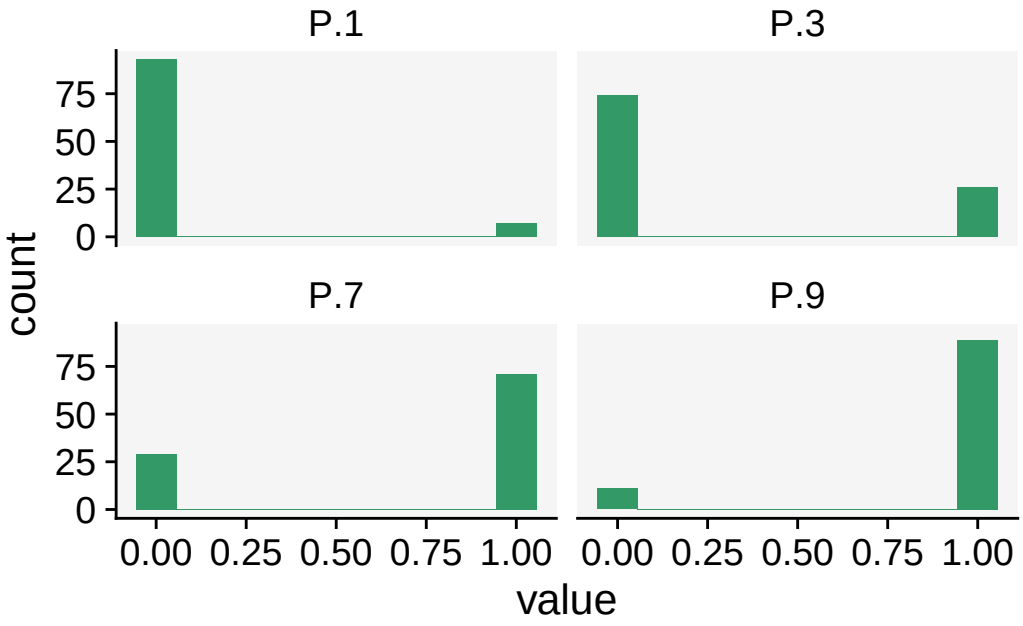


Figure 2.4: Binomial distribution to describe binary data

For example, if we're studying a plant disease, we could express disease incidence as the proportion of plants that are newly diseased during a given time period. Similarly, disease severity could be expressed as the proportion of each plant's organ area that is affected by the disease. These proportions are ratio variables, as they can take on any value between 0 and 1, and ratios of these variables are meaningful.

The Beta distribution is a probability distribution that is defined between 0 and 1, which makes it ideal for modeling data that represents proportions. It's a flexible distribution, as its shape can take many forms depending on the values of its two parameters, often denoted as alpha and beta.

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.

```r
beta_data |>
  pivot_longer(1:2, names_to = "P",
               values_to = "value") |>
  ggplot(aes(value)) +
  geom_histogram(fill = "#339966",
                 color = "white",
                 bins = 15) +
  scale_x_continuous(limits = c(0, 1)) +
  facet_wrap( ~ P)+
  theme_r4pde()
```

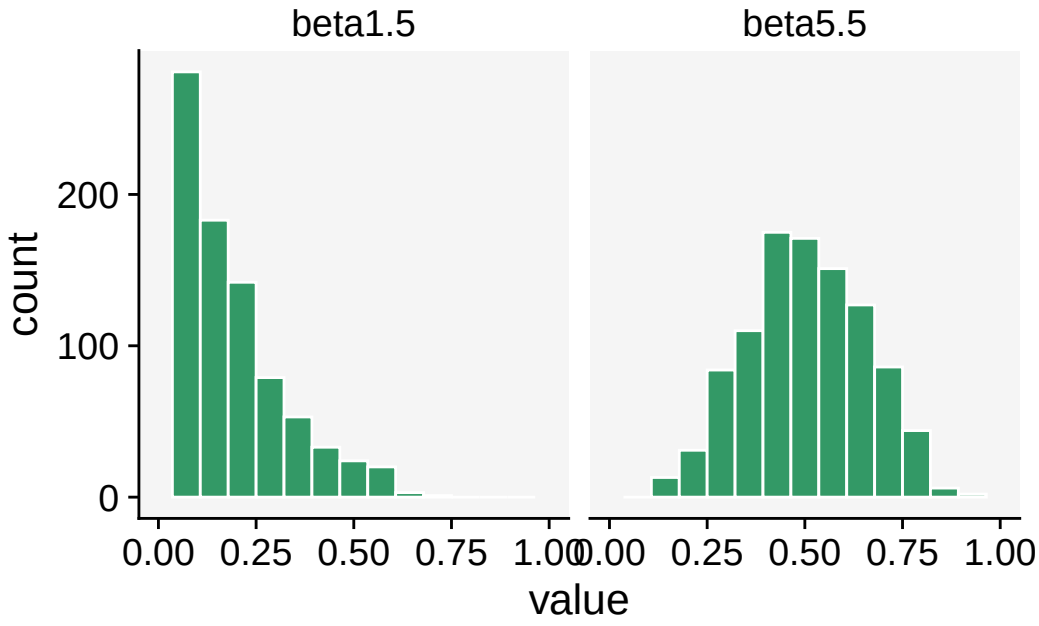


Figure 2.5: Binomial distribution to describe proportion data

2.4.3 Beta-binomial distribution

The Beta-Binomial distribution is a mixture of the Binomial distribution with the Beta distribution acting as a prior on the probability parameter of the binomial. Disease probabilities can vary across trials due to a number of unobserved or unmeasured factors. This variability can result in overdispersion, a phenomenon where the observed variance in the data is greater than what the binomial distribution expects.

This is where the Beta-Binomial distribution comes in handy. By combining the Beta dis-

tribution's flexibility in modeling probabilities with the Binomial distribution's discrete event modeling, it provides an extra layer of variability to account for overdispersion. The Beta-Binomial distribution treats the probability of success (disease occurrence in this context) as a random variable itself, following a Beta distribution. This means the probability can vary from trial to trial.

Therefore, when we observe data that shows more variance than the Beta distribution can account for, or when we believe there are underlying factors causing variability in the probability of disease occurrence, the Beta-Binomial distribution is a more appropriate model. It captures both the variability in success probability as well as the occurrence of the discrete event (disease incidence).

When combined with the Binomial distribution, which handles discrete events (e.g. whether an individual is diseased or not), the Beta-Binomial distribution allows us to make probabilistic predictions about these events. For example, based on prior data (the Beta distribution), we can estimate the likelihood of a particular individual being diseased (the Binomial distribution).

In R, the `rBetaBin` function of the *FlexReg* package generates random values from the beta-binomial distribution. The arguments of the function are `n`, or the number of values to generate; if length(`n`) > 1, the length is taken to be the number required. `size` is he total number of trials. `mu` is the mean parameter. It must lie in (0, 1). `theta` is the overdispersion parameter. It must lie in (0, 1). `phi` the precision parameter. It is an alternative way to specify the theta parameter. It must be a positive real value.

```
library(FlexReg)
betabin3.6 <- rBetaBin(n = 100, size = 40, mu = .3, theta = .6)
betabin7.3 <- rBetaBin(n = 100, size = 40, mu = .7, theta = .3)
betabin_data <- data.frame(betabin3.6, betabin7.3)
```

```
betabin_data |>
  pivot_longer(1:2, names_to = "P",
               values_to = "value") |>
  ggplot(aes(value)) +
  geom_histogram(fill = "#339966",
                 color = "white",
                 bins = 15) +
  facet_wrap( ~ P) +
  theme_r4pde()
```

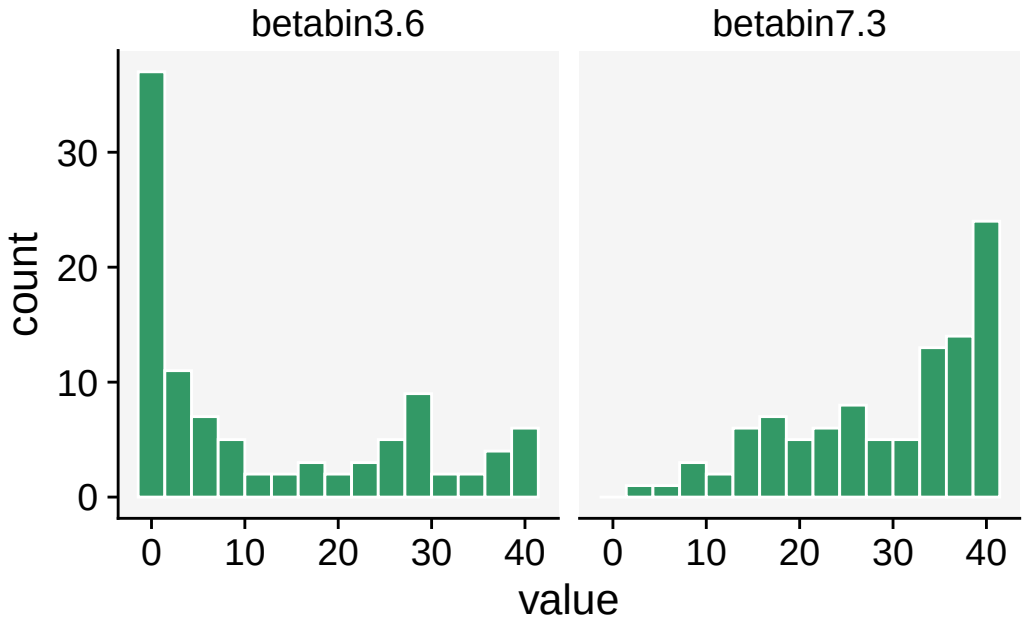


Figure 2.6: Beta-binomial distribution to describe proportion data

2.4.4 Poisson distribution

When conducting studies in epidemiology, specifically plant diseases, researchers often collect data on the number of diseased plants, infected plant parts, or individual symptoms, such as lesions. These variables are counted in whole numbers - 1, 2, 3, etc., making them discrete variables. Discrete variables contrast with continuous variables that can take any value within a defined range and can include fractions or decimals. In addition to being discrete, these variables are also non-negative, meaning they cannot take negative values. After all, you can't have a negative number of diseased plants or lesions. Given these characteristics, a suitable distribution to model such data is the Poisson distribution. This distribution is particularly suitable for counting the number of times an event occurs in a given time or space.

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.

```
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 = "#339966",
                 color = "white",
                 bins = 15) +
  facet_wrap( ~ P) +
  theme_r4pde()
```

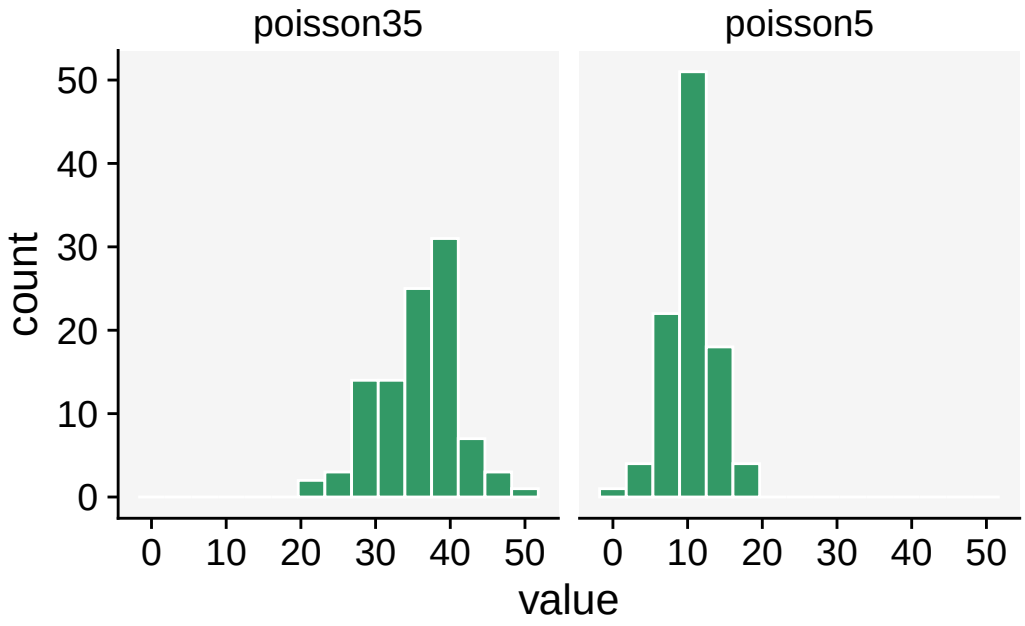


Figure 2.7: Poisson distribution to describe count data

2.4.5 Negative binomial distribution

While the Poisson distribution is indeed suitable for modeling count data, it assumes that the mean and variance of the data are equal. However, in real-world scenarios, especially in epidemiology, it is common to encounter overdispersed data - where the variance is greater than the mean. This could occur, for instance, if there's greater variability in disease incidence across different plant populations than would be expected under the Poisson assumption.

In such cases, the Negative Binomial distribution is a better alternative. The Negative Binomial distribution is a discrete probability distribution that models the number of successes

in a sequence of independent and identically distributed Bernoulli trials before a specified (non-random) number of failures occurs.

One of the key features of the Negative Binomial distribution is its ability to handle overdispersion. Unlike the Poisson distribution, which has one parameter (lambda, representing the mean and variance), the Negative Binomial distribution has two parameters. One parameter is the mean, but the other (often denoted as 'size' or 'shape') governs the variance independently, allowing it to be larger than the mean if necessary. Thus, it provides greater flexibility than the Poisson distribution for modeling count data and can lead to more accurate results when overdispersion is present.

In R, we can use the `rnbinom()` function to generate random variates from a Negative Binomial distribution. This function requires the number of observations (`n`), the target for the number of successful trials (`size`), and the probability of each success (`prob`).

Here's an example:

```
# Generate 100 random variables from a Negative Binomial distribution
negbin14.6 <- rnbinom(n = 100, size = 14, prob = 0.6)
negbin50.6 <- rnbinom(n = 100, size = 50, prob = 0.6)
negbin_data <- data.frame(negbin14.6, negbin50.6)
```

```
negbin_data |>
  pivot_longer(1:2, names_to = "P",
               values_to = "value") |>
  ggplot(aes(value)) +
  geom_histogram(fill = "#339966",
                 color = "white", bins = 15) +
  facet_wrap( ~ P) +
  theme_r4pde()
```

2.4.6 Gamma distribution

In plant disease epidemiology and other fields of study, we may often encounter continuous variables - these are variables that can take on any value within a given range, including both whole numbers and fractions. An example of a continuous variable in this context is lesion size, which can theoretically be any non-negative value.

Often, researchers use the normal (Gaussian) distribution to model such continuous variables. The normal distribution is symmetric, bell-shaped, and is fully described by its mean and standard deviation. However, a fundamental characteristic of the normal distribution is that it extends from negative infinity to positive infinity. While this is not a problem for many

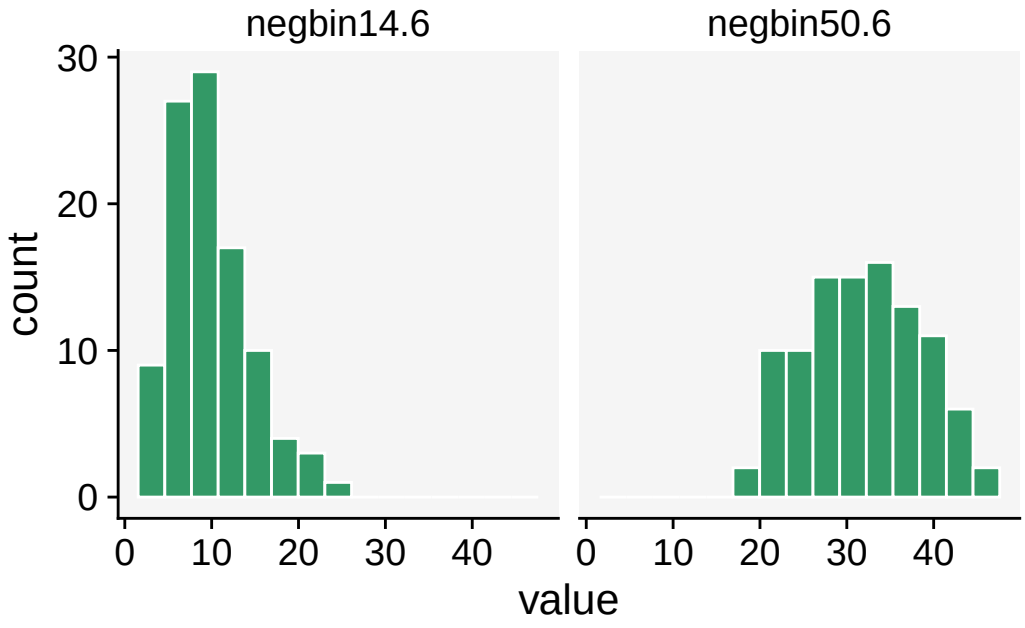


Figure 2.8: Negative binomial distribution to describe overdispersed count data

applications, it becomes an issue when the variable being modeled cannot take on negative values - like the size of a lesion.

This is where the Gamma distribution can be a good alternative. The Gamma distribution is a two-parameter family of continuous probability distributions, which does not include negative values, making it an appropriate choice for modeling variables like lesion sizes. While it might seem a bit more complicated due to its two parameters, this also allows it a greater flexibility in terms of the variety of shapes and behaviors it can describe. The Gamma distribution is often used in various scientific disciplines, including queuing models, climatology, financial services, and of course, epidemiology. Its main parameters are the shape and scale (or alternatively shape and rate), which control the shape, spread and location of the distribution.

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)
```

```r
gamma_data |>
  pivot_longer(1:2, names_to = "P",
               values_to = "value") |>
  ggplot(aes(value)) +
  geom_histogram(fill = "#339966",
                 color = "white",
                 bins = 15) +
  ylim(0, max(gamma_data$gamma35)) +
  facet_wrap( ~ P) +
  theme_r4pde()
```

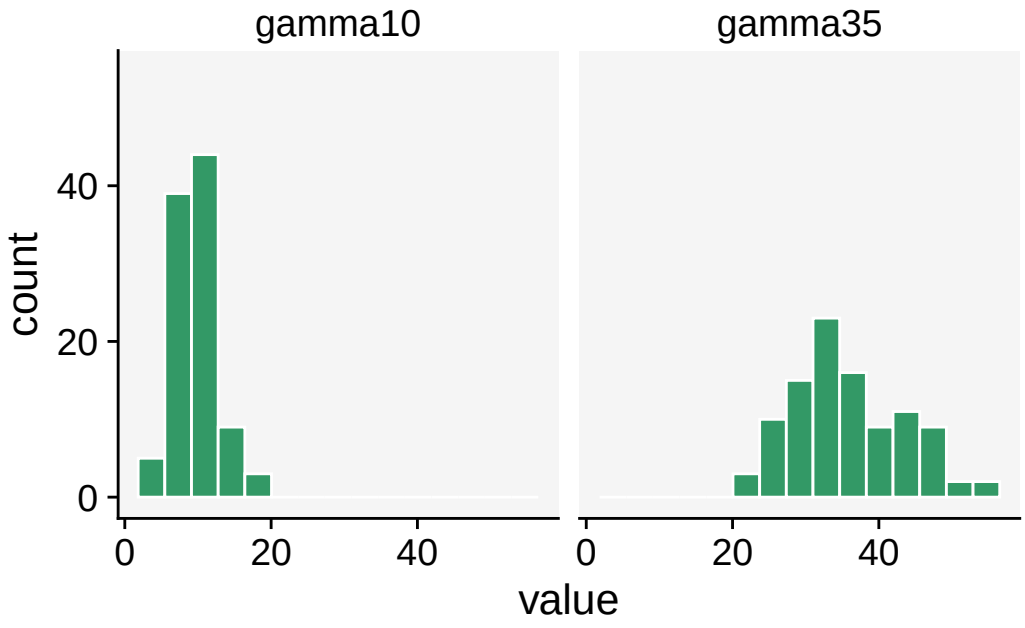


Figure 2.9: Gamma distribution to describe continuous data

2.4.7 Simulating ordinal data

Ordinal data is a statistical data type consisting of numerical scores that fall into a set of categories which are ordered in a meaningful way. This can include survey responses (e.g., strongly disagree to strongly agree), levels of achievement (e.g., poor, average, good, excellent), or, in the case of plant disease, disease severity scales (e.g., 0 to 5, where 0 represents a healthy plant and 5 represents a plant with severe symptoms).

When working with ordinal data, we often need to make assumptions about the distribution

of the data. However, unlike continuous data which might be modeled by a normal or Gamma distribution, or count data which might be modeled by a Poisson distribution, ordinal data is discrete and has a clear order but the distances between the categories are not necessarily equal or known. This makes the modeling process slightly different.

We can use the `sample()` function and define the probability associated with each rank. Let's generate 30 units with a distinct ordinal score. In the first situation, the higher probabilities (0.5) are for scores 4 and 5 and lower (0.1) for scores 0 and 1, and in the second situation is the converse.

```
ordinal1 <- sample(0:5, 30, replace = TRUE, prob = c(0.1, 0.1, 0.2, 0.2, 0.5, 0.5))
ordinal2 <- sample(0:5, 30, replace = TRUE, prob = c(0.5, 0.5, 0.2, 0.2, 0.1, 0.1))
ordinal_data <- data.frame(ordinal1, ordinal2)
```

```
ordinal_data |>
  pivot_longer(1:2, names_to = "P",
               values_to = "value") |>
  ggplot(aes(value)) +
  geom_histogram(fill = "#339966",
                 color = "white",
                 bins = 6) +
  facet_wrap( ~ P) +
  theme_r4pde()
```

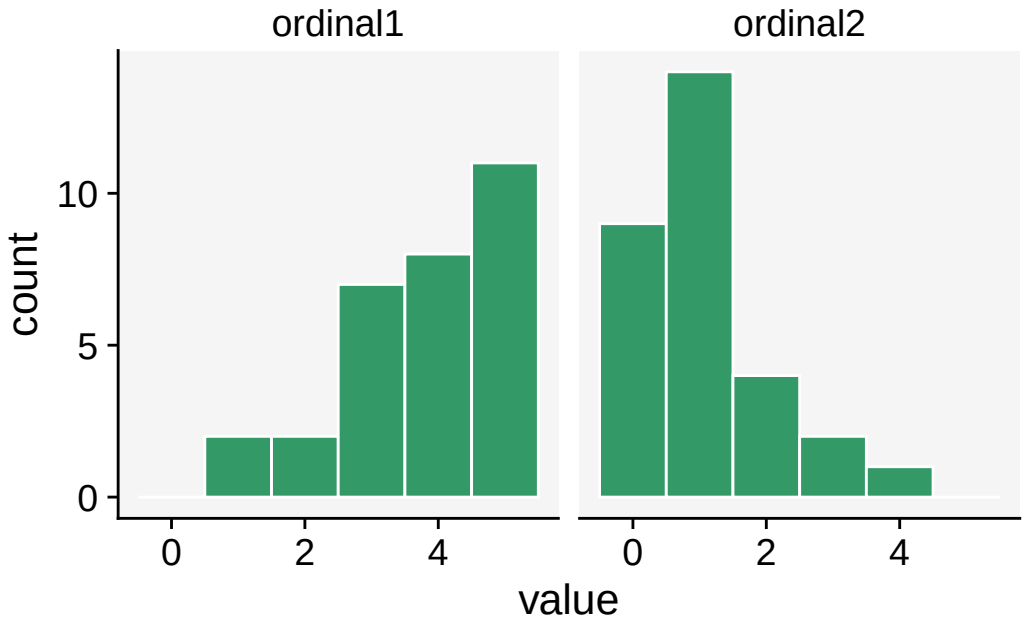


Figure 2.10: Sampling of ordinal data

3 Ordinal scales

3.1 Ordinal scales

Ordinal scales are organized as rank-ordered numeric classes, with a finite number of such classes. The utilization of ordinal scales is often due to their convenience and speed of rating (Madden et al. 2007d). In plant pathological research, there are two commonly used types of ordinal scales: quantitative and qualitative (Chiang and Bock 2021).

3.1.1 Quantitative ordinal

In the quantitative ordinal scale, each score signifies a defined interval of the percentage scale. The most renowned quantitative ordinal scale is the Horsfall-Barratt (HB) scale, which was developed in the early 1940s when the science of plant pathology was transitioning towards more quantitative methodologies (Hebert 1982). The HB scale partitions the percentage scale into twelve successive, logarithmic-based intervals of severity ranging from 0 to 100%. The intervals increase in size from 0 to 50% and decrease from 50 to 100%.

> ⚠ Controversy of the H-B scale
>
> The divisions of the H-B scale were established on two assumptions. The first was the logarithmic relationship between the intensity of a stimulus and the subsequent sensation. The second was the propensity of a rater to focus on smaller objects when observing objects of two colors (Madden et al. 2007d). This foundation is based on the so-called Weber-Fechner law. However, there is limited experimental evidence supporting these assumptions. Current evidence indicates a linear relationship, rather than a logarithmic one, between visually estimated and actual severity (Nutter and Esker 2006). Additionally, these authors demonstrated that raters more accurately discriminated disease severity between 25% and 50% than what the H-B scale allowed. New scale structures have been proposed to address the issues associated with the H-B scale (Liu et al. 2019; Chiang et al. 2014). The Chiang scale follows a linear relationship with the percentage area diseased at severities greater than 10% (class 6 on the scale).

Let's input the HB scale data and store as a data frame in R so we can prepare a table and a plot.

```
HB <- tibble::tribble(
  ~ordinal, ~'range', ~midpoint,
  0,          '0',    0,
  1,    '0+ to 3',  1.5,
  2,    '3+ to 6',  4.5,
  3,   '6+ to 12',  9.0,
  4,  '12+ to 25', 18.5,
  5,  '25+ to 50', 37.5,
  6,  '50+ to 75', 62.5,
  7,  '75+ to 88', 81.5,
  8,  '88+ to 94', 91.0,
  9,  '94+ to 97', 95.5,
  10,'97+ to 100', 98.5,
  11,      '100',   100
  )
knitr::kable(HB, align = "c")
```

Table 3.1: The Horsfal-Barrat quantitative ordinal scale used as a tool for assessing plant disease severity

ordinal	range	midpoint
0	0	0.0
1	0+ to 3	1.5
2	3+ to 6	4.5
3	6+ to 12	9.0
4	12+ to 25	18.5
5	25+ to 50	37.5
6	50+ to 75	62.5
7	75+ to 88	81.5
8	88+ to 94	91.0
9	94+ to 97	95.5
10	97+ to 100	98.5
11	100	100.0

Let's visualize the different sizes of the percent interval encompassing each score.

```
library(tidyverse)
library(r4pde)
HB |>
  ggplot(aes(midpoint, ordinal))+
```

```
  geom_point(size =2)+
  geom_line()+
  scale_x_continuous(breaks = c(0, 3, 6, 12, 25, 50, 75, 88, 94, 97))+
  scale_y_continuous(breaks = c(1:12))+
  geom_vline(aes(xintercept = 3), linetype = 2)+
  geom_vline(aes(xintercept = 6), linetype = 2)+
  geom_vline(aes(xintercept = 12), linetype = 2)+
  geom_vline(aes(xintercept = 25), linetype = 2)+
  geom_vline(aes(xintercept = 50), linetype = 2)+
  geom_vline(aes(xintercept = 75), linetype = 2)+
  geom_vline(aes(xintercept = 88), linetype = 2)+
  geom_vline(aes(xintercept = 94), linetype = 2)+
  geom_vline(aes(xintercept = 97), linetype = 2)+
  labs(x = "Percent severity", y = "HB score")+
  theme_r4pde()
```

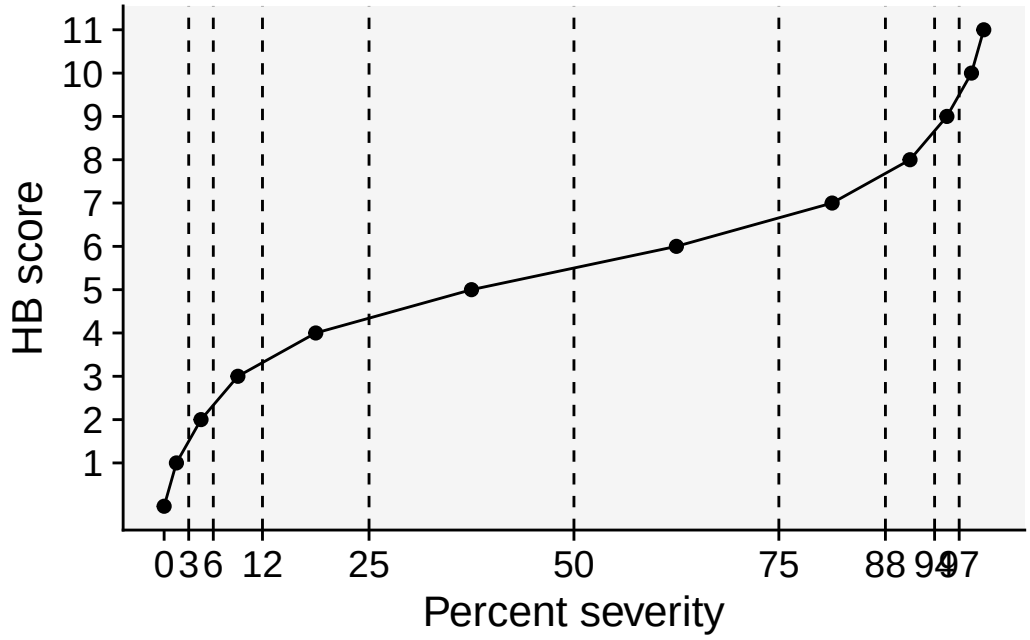


Figure 3.1: Ordinal scores of the Horsfal-Barrat scale

We can repeat those procedures to visualize the Chiang scale.

```
chiang <- tibble::tribble(
  ~ordinal, ~'range', ~midpoint,
```

```
  0,          '0',     0,
  1,  '0+ to 0.1',  0.05,
  2,'0.1+ to 0.5',   0.3,
  3,  '0.5+ to 1',  0.75,
  4,    '1+ to 2',   1.5,
  5,    '2+ to 5',     3,
  6,   '5+ to 10',   7.5,
  7,  '10+ to 20',    15,
  8,  '20+ to 30',    25,
  9,  '30+ to 40',    35,
  10, '40+ to 50',    45,
  11, '50+ to 60',    55,
  12, '60+ to 70',    65,
  13, '70+ to 80',    75,
  14, '80+ to 90',    85,
  15,'90+ to 100',   95
  )
knitr::kable(chiang, align = "c")
```

Table 3.2: The Chiang quantitative ordinal scale used as a tool for assessing plant disease severity

ordinal	range	midpoint
0	0	0.00
1	0+ to 0.1	0.05
2	0.1+ to 0.5	0.30
3	0.5+ to 1	0.75
4	1+ to 2	1.50
5	2+ to 5	3.00
6	5+ to 10	7.50
7	10+ to 20	15.00
8	20+ to 30	25.00
9	30+ to 40	35.00
10	40+ to 50	45.00
11	50+ to 60	55.00
12	60+ to 70	65.00
13	70+ to 80	75.00
14	80+ to 90	85.00
15	90+ to 100	95.00

```
chiang |>
  ggplot(aes(midpoint, ordinal))+
  geom_point(size =2)+
  geom_line()+
  scale_y_continuous(breaks = c(0:15))+
  scale_x_continuous(breaks = c(0, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100))+
  geom_vline(aes(xintercept = 0), linetype = 2)+
  geom_vline(aes(xintercept = 0.1), linetype = 2)+
  geom_vline(aes(xintercept = 0.5), linetype = 2)+
  geom_vline(aes(xintercept = 1), linetype = 2)+
  geom_vline(aes(xintercept = 2), linetype = 2)+
  geom_vline(aes(xintercept = 5), linetype = 2)+
  geom_vline(aes(xintercept = 10), linetype = 2)+
  geom_vline(aes(xintercept = 20), linetype = 2)+
  geom_vline(aes(xintercept = 30), linetype = 2)+
   geom_vline(aes(xintercept = 40), linetype = 2)+
   geom_vline(aes(xintercept = 50), linetype = 2)+
   geom_vline(aes(xintercept = 60), linetype = 2)+
   geom_vline(aes(xintercept = 70), linetype = 2)+
   geom_vline(aes(xintercept = 80), linetype = 2)+
   geom_vline(aes(xintercept = 90), linetype = 2)+
   geom_vline(aes(xintercept = 100), linetype = 2)+
  labs(x = "Percent severity", y = "Chiang score")+
  theme_r4pde()
```

3.1.2 Qualitative ordinal

In the qualitative ordinal scale, each class provides a description of the symptoms. An example is the ordinal 0-3 scale for rating eyespot of wheat developed by (Scott and Hollins 1974).

Table 3.3: Ordinal scale for rating eyespot of wheat (Scott and Hollins 1974)

Class	Description
0	uninfected
1	slight eyespot (or or more small lesion occupying in total less than half of the circumference of the stem)
2	moderate eyespot (one or more lesions occupying at least half the circumference of the stem)
3	severe eyespot (stem completely girdled by lesions; tissue softened so that lodging would really occur)

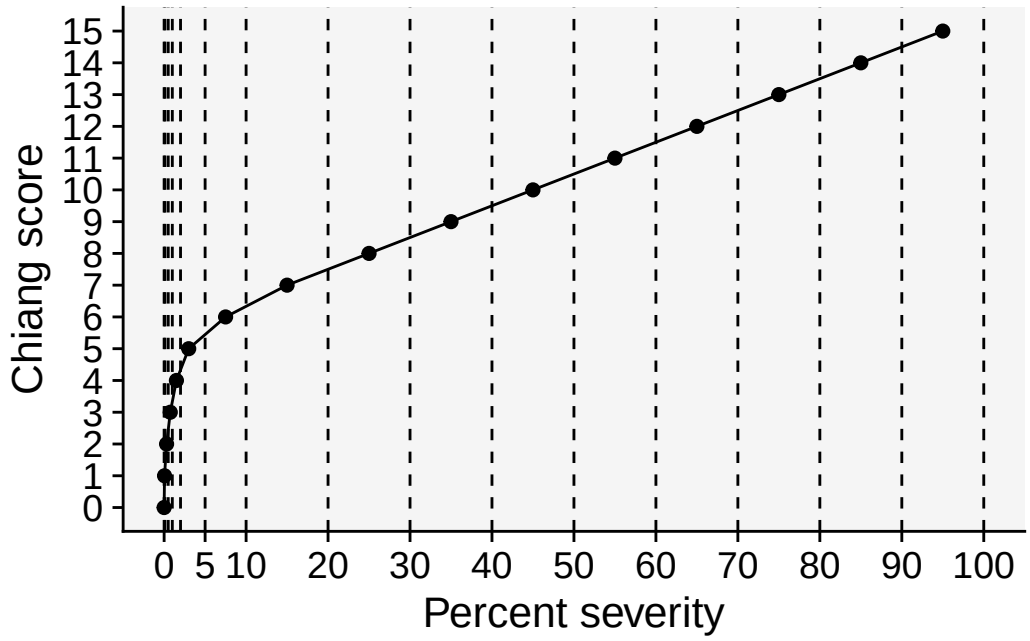


Figure 3.2: Ordinal scores of the Chiang scale

3.2 Disease severity index (DSI)

Usually, when quantitative or qualitative ordinal scales are used, the scores are transformed into an index on a percentage basis, such as the disease severity index (DSI) which is used in data analysis. The DSI is a single number that summarizes a large amount of information on disease severity (Chester 1950). The formula for a DSI (%) can be written as follows:

$DSI = \frac{\sum(class\ freq. \quad score\ of\ class)}{total\ n \quad maximal\ class}\ 100$

The `DSI()` and `DSI2()` are part of the *r4pde* package. Let's see how each function works.

The `DSI()` allows to automate the calculation of the disease severity index (DSI) in a series of units (e.g. leaves) that are further classified according to ordinal scores. The function requires three arguments:

- unit = the vector of the number of each unit
- class = the vector of the scores for the units
- max = the maximum value of the scale

Let's create a toy data set composed of 12 units where each received an ordinal score. The vectors were arranged as a data frame named scores.

```
unit <- c(1:12)
class <- c(2,3,1,1,3,4,5,0,2,5,2,1)
ratings <- data.frame(unit, class)
knitr::kable(ratings)
```

unit	class
1	2
2	3
3	1
4	1
5	3
6	4
7	5
8	0
9	2
10	5
11	2
12	1

The ordinal score used in this example has 6 as the maximum score. The function returns the DSI value.

```
library(r4pde)
DSI(ratings$unit, ratings$class, 6)
```

```
[1] 40.27778
```

Let's now deal with a situation of multiple plots (five replicates) where a fixed number of 12 samples were taken and assessed using an ordinal score. Let's input the data using the `tribble()` function. Note that the data is in the wide format.

```
exp <- tibble::tribble(
  ~rep, ~`1`, ~`2`, ~`3`, ~`4`, ~`5`, ~`6`, ~`7`, ~`8`, ~`9`, ~`10`, ~`11`,~`12`,
  1, 2, 3, 1, 1, 3, 4, 5, 0, 2, 5, 2, 1,
  2, 3, 4, 4, 6, 5, 4, 4, 0, 2, 1, 1, 5,
  3, 5, 6, 6, 5, 4, 2, 0, 0, 0, 0, 2, 0,
  4, 5, 6, 0, 0, 0, 3, 3, 2, 1, 0, 2, 3,
  5, 0, 0, 0, 0, 2, 3, 2, 5, 6, 2, 1, 0,
)
```

```
knitr::kable(exp)
```

rep	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	1	1	3	4	5	0	2	5	2	1
2	3	4	4	6	5	4	4	0	2	1	1	5
3	5	6	6	5	4	2	0	0	0	0	2	0
4	5	6	0	0	0	3	3	2	1	0	2	3
5	0	0	0	0	2	3	2	5	6	2	1	0

After reshaping the data to the long format, we can calculate the DSI for each plot/replicate as follows:

```
res <- exp |>
  pivot_longer(2:13, names_to = "unit", values_to = "class") |>
  group_by(rep) |>
  summarise(DSI = DSI(unit, class, 6))
```

And here we have the results of the DSI for each replicate.

```
knitr::kable(res, align = "c")
```

rep	DSI
1	40.27778
2	54.16667
3	41.66667
4	34.72222
5	29.16667

Now our data set is organized as the frequency of each class as follows:

```
ratings2 <- ratings |>
  dplyr::count(class)

ratings2
```

```
  class n
1     0 1
2     1 3
```

```
3     2 3
4     3 2
5     4 1
6     5 2
```

Now we can apply the `DSI2()` function. The function requires three arguments:

- class = the number of the respective class
- freq = the frequency of the class
- max = the maximum value of the scale

```
library(r4pde)
DSI2(ratings2$class, ratings2$n, 6)
```

```
[1] 40.27778
```

4 Image analysis

4.1 The actual severity measure

Among the various methods to express plant disease severity, the percent area affected (or symptomatic) by the disease is one of the most common, especially when dealing with diseases that affect leaves. In order to evaluate whether visual estimates of plant disease severity are sufficiently accurate (as discussed in the previous chapter), we require the actual severity values. These are also essential when creating Standard Area Diagrams (SADs), which are diagrammatic representations of severity values used as a reference either before or during visual assessment to standardize and produce more accurate results across different raters (Del Ponte et al. 2017).

The actual severity values are typically approximated using image analysis, wherein the image is segmented, and each pixel is categorized into one of three classes:

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

The ratio of the diseased area to the total area of the unit (e.g., the entire plant organ or section of the image) yields the proportion of the diseased area, or the percent area affected (when multiplied by 100). Researchers have employed various proprietary or open-source software to determine the actual severity, as documented in a review on Standard Area Diagrams (Del Ponte et al. 2017).

In this section, we will utilize the `measure_disease()` function from the {pliman} (Plant IMage ANalysis) R package (Olivoto 2022), and its variations, to measure the percent area affected. The package was compared with other software for determining plant disease severity across five different plant diseases and was shown to produce accurate results in most cases (Olivoto et al. 2022).

There are essentially two methods to measure severity. The first is predicated on image palettes that define each class of the image. The second relies on RGB-based indices (Alves et al. 2021). Let's explore the first method, as well as an interactive approach to setting color palettes.

4.2 Image palettes

The most crucial step is the initial one, where the user needs to correctly define the color palettes for each class. In pliman, the palettes can be separate images representing each of the three classes: background (b), symptomatic (s), and healthy (h).

The reference image palettes can be constructed by manually sampling small areas of the image and creating a composite image. As expected, the results may vary depending on how these areas are selected. A study that validated pliman for determining disease severity demonstrated the effect of different palettes prepared independently by three researchers, in which the composite palette (combining the three users) was superior (Olivoto et al. 2022). During the calibration of the palettes, examining the processed masks is crucial to create reference palettes that are the most representative of the respective class.

In this example, I manually selected and pasted several sections of images representing each class from a few leaves into a Google slide. Once the image palette was ready, I exported each one as a separate PNG image file (JPG also works). These files 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 need to import them into the environment, using `image_import()` function 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)
```

An alternative way to set the palettes is to use the `pick_palette()` function. It allows to manually pick the colors for each class by clicking on top of the imported image. We can use one of the original images or a composite images with portions of several leaves. Let's use here one of the original images and pick the background colors and assigned to `b` vector.

```
img <- image_import("imgs/originals/img5.png")
b <- pick_palette(img)
```

The original image is displayed and the user needs to click on the background colors to select the pixels. A message will be displayed as follows:

> Use the first mouse button to pick up points in the plot. Press Esc to exit.

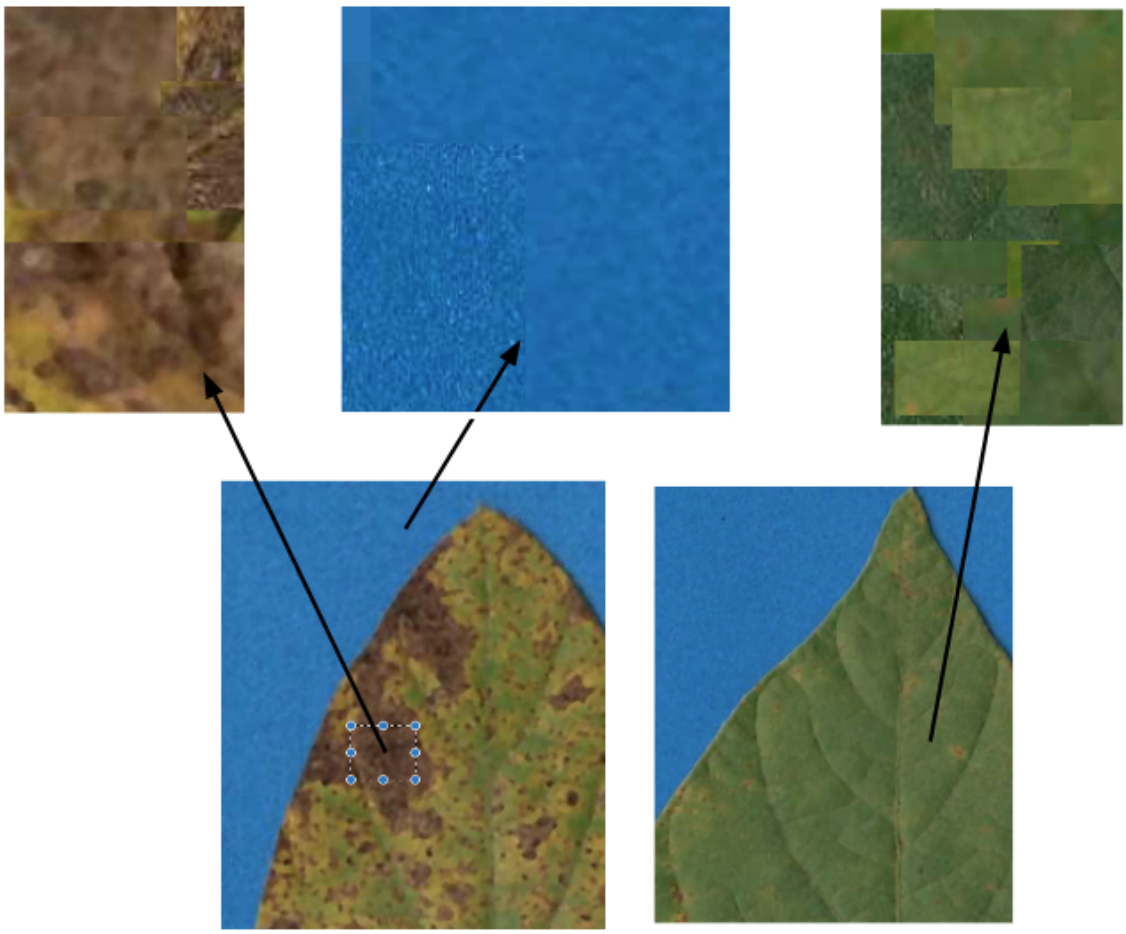


Figure 4.1: Preparation of image palettes by manually sampling fraction of the images that represent background, heatlhy leaf and lesions

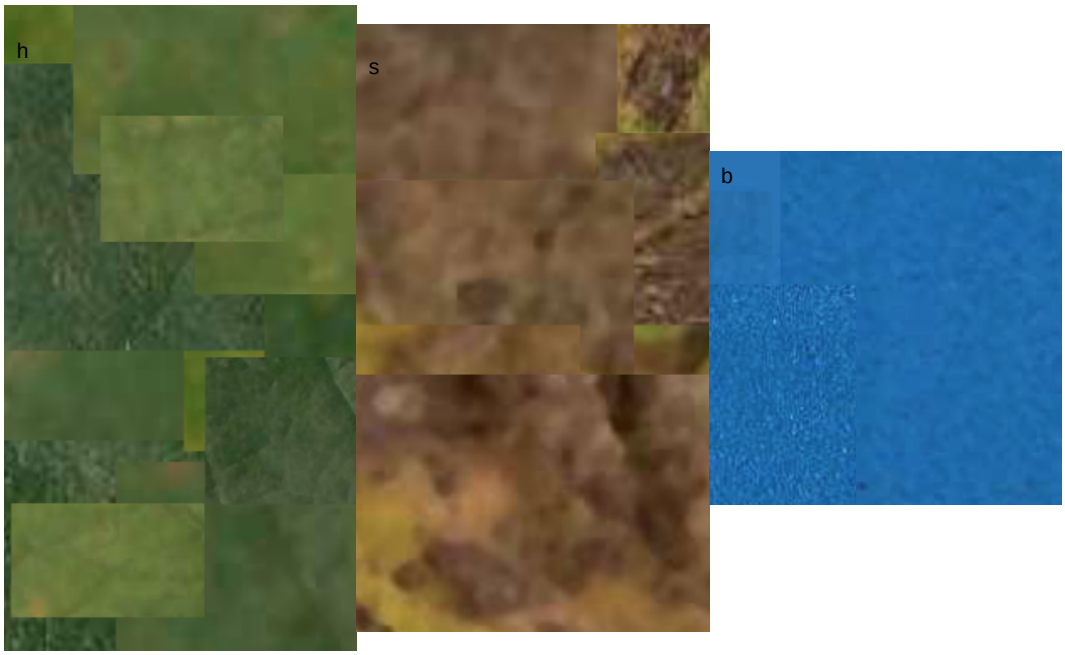


Figure 4.2: Image palettes created to segment images into background, sypomtoms and healthy area of the image

After pressing ESC, the image palette for the background is constructed and can be displayed.

```
image_combine(b)
```

Now, we can proceed and pick the colors for the other categories following the same logic.

```
# Symptoms
s <- pick_palette(img)

# healthy
h <- pick_palette(img)
```

Figure 4.3: Soybean rust leaf with small red dots at the upper leaf portion that indicate which colors were selected for the background

Figure 4.4: Image generating after picking the palette colors for the background of the leaf

4.3 Measuring severity

4.3.1 Single image

4.3.1.1 Using color palettes

To determine severity in a single image (e.g. 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.

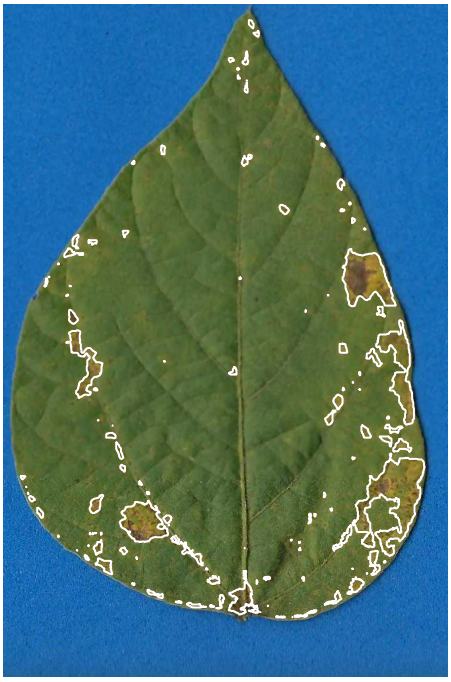
```
img <- image_import("imgs/originals/img46.png")
image_combine(img)
```

Figure 4.5: Imported image for further analysis

Now the engaging part starts with the `measure_disease()` function. Four arguments are required when using the reference image palettes: the image representing the target image and the three images of the color palettes. As the author of the package states, "pliman will

take care of all the details!" The severity is the value displayed under 'symptomatic' in the output.

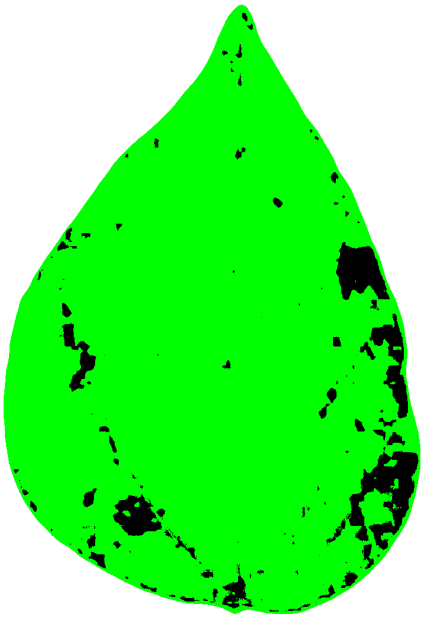
```
set.seed(123)
measure_disease(
  img = img,
  img_healthy = h,
  img_symptoms = s,
  img_background = b
)
```



If we want to show the mask with two colors instead of the original, we can set to FALSE two "show_" arguments:

```
set.seed(123)
measure_disease(
  img = img,
  img_healthy = h,
  img_symptoms = s,
  img_background = b,
  show_contour = FALSE,
  show_original = FALSE
```

```
)
```



4.3.2 Multiple images

Measuring severity in single images is indeed engaging, but we often deal with multiple images, not just one. Using the above procedure to process each image individually would be time-consuming and potentially tedious.

To automate the process, {pliman} offers a batch processing approach. Instead of using the `img` argument, one can use the `pattern` argument and define the prefix of the image names. Moreover, we also need to specify the directory where the original files are located.

If the user wants to save the processed masks, they should set the `save_image` argument to TRUE and also specify the directory where the images will be saved. Here's an example of how to process 10 images of soybean rust symptoms. The output 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,
  verbose = FALSE,
  plot = FALSE
)
```

```
Done!

Elapsed time: 00:00:22
```

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

```
     img  healthy symptomatic
1  img11 70.79655  29.2034481
2  img35 46.94177  53.0582346
3  img37 60.47440  39.5256013
4  img38 79.14060  20.8594011
5  img46 93.14958   6.8504220
6   img5 20.53175  79.4682534
7  img63 97.15669   2.8433141
8  img67 99.83720   0.1627959
9  img70 35.56684  64.4331583
10 img75 93.04453   6.9554686
```

When the argument `save_image` is 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.

4.3.2.1 More than a target per image

{pliman} offers a custom function to estimate the severity in multiple targets (e.g. leaf) per image. This is convenient to decrease the time when scanning the specimens, for example. Let's combine three soybean rust leaves into a single image and import it for processing. We will further set the `index_lb` (leaf background), `save_image` to `TRUE` and inform the directory for the processed images using `dir_processed`.

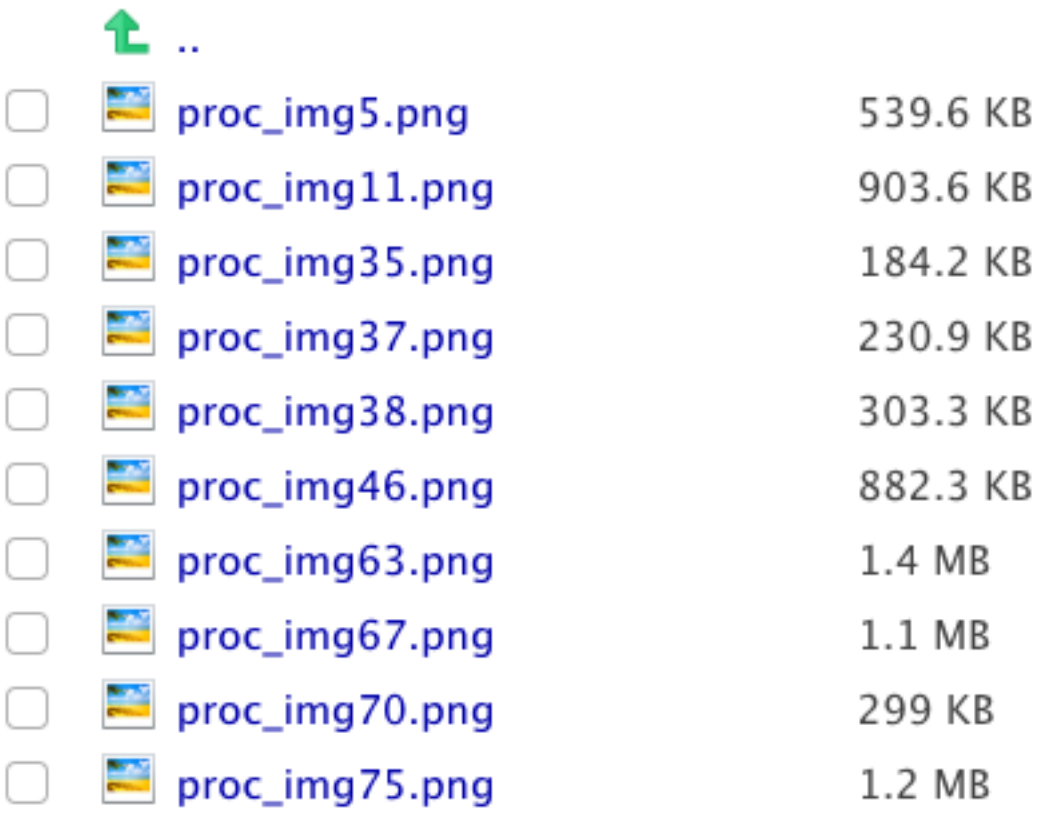


Figure 4.6: Images created by pliman and exported to a specific folder

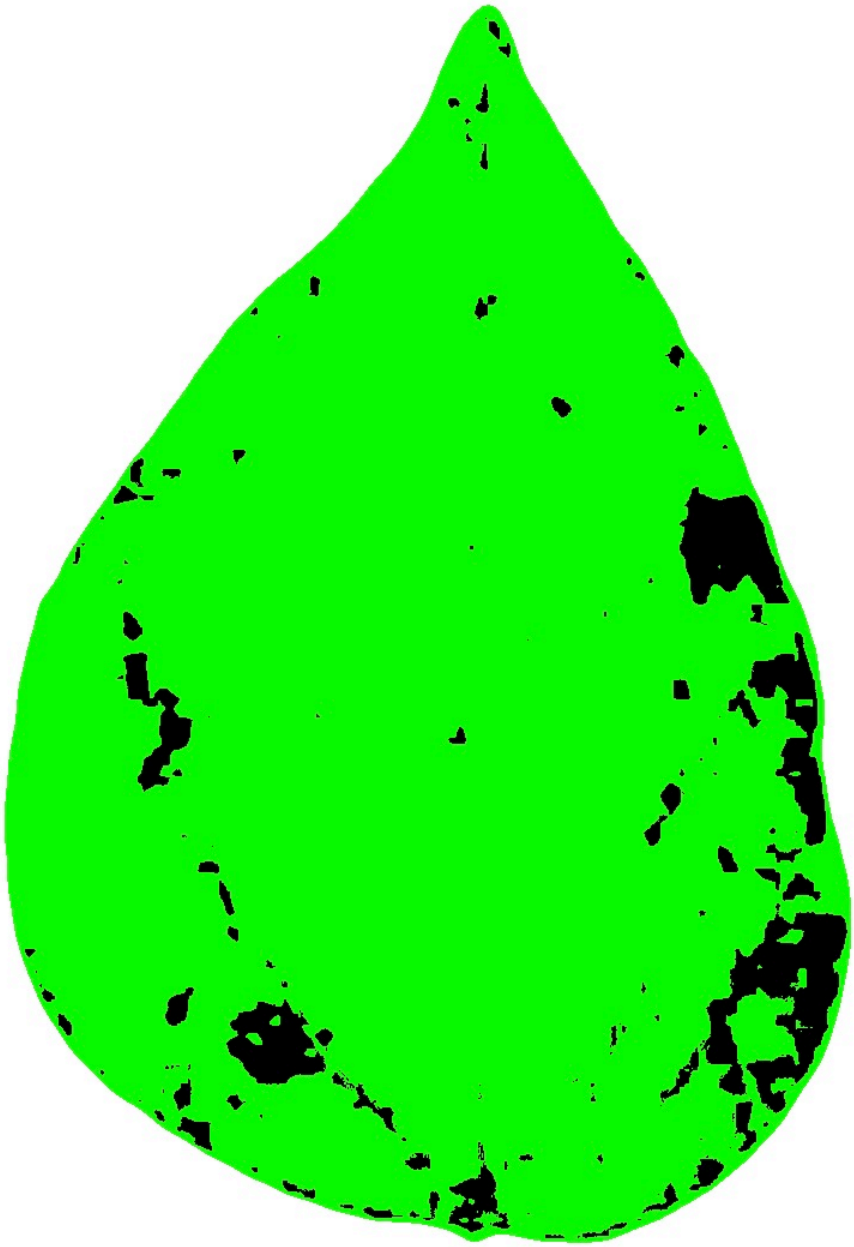
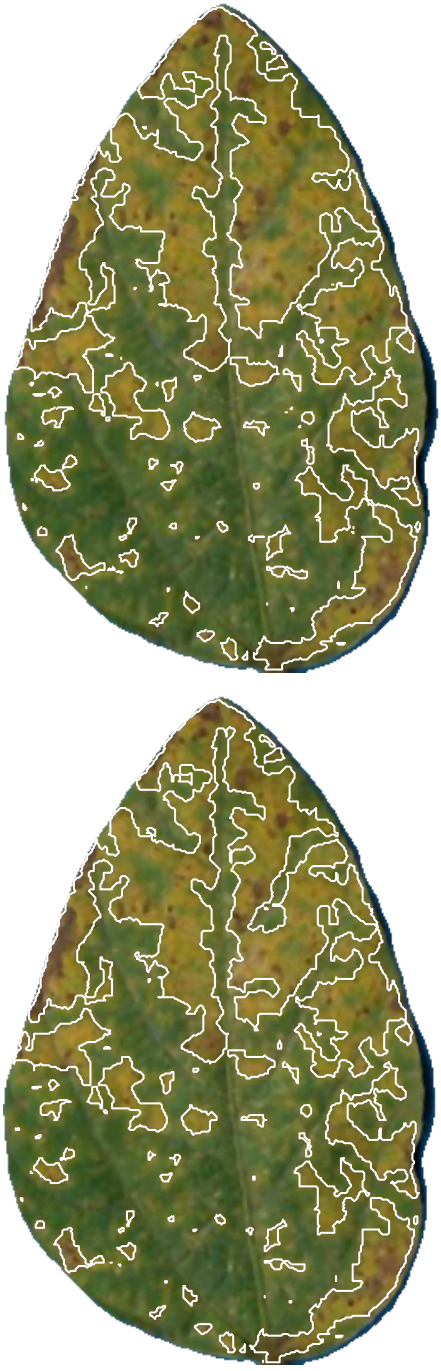


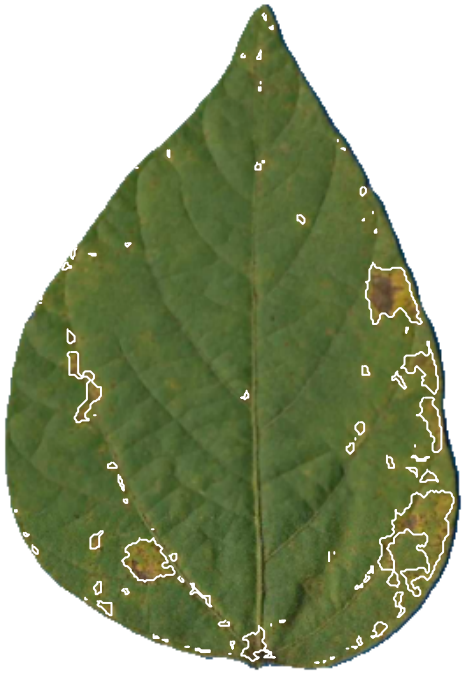
Figure 4.7: Figure created by pliman after batch processing to segment the images and calculate percent area covered by symptoms. The symptomatic area is delinated in the image.

```
img2 <- image_import("imgs/soybean_three.png")
image_combine(img2)
```

Figure 4.8: Imported image with multiple targets in a single image for further analysis using measure_disease_byl() function of the {pliman} package

```
pliman2 <- measure_disease_byl(img = img2,
                          index_lb = b,
                          img_healthy = h,
                          img_symptoms = s,
                          save_image = TRUE,
                          dir_processed = "imgs/proc")
```





```
  pliman2$severity
```

```
  img leaf  healthy symptomatic
1 img    1 59.11737   40.882632
2 img    2 61.36616   38.633841
3 img    3 93.21756    6.782436
```

The original image is splited and the individual new images are saved in the proc folder.

4.4 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)
library(r4pde)
quant <- tribble(
  ~img, ~actual,
   "img5",     75,
```

Figure 4.9: Individual images of the soybean leaves after processed using the measure_disease_byl function of the {pliman} package.

```
  "img11",     24,
  "img35",     52,
  "img37",     38,
  "img38",     17,
  "img46",      7,
  "img63",    2.5,
  "img67",   0.25,
  "img70",     67,
  "img75",     10
  )
```

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

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

```
Joining with `by = join_by(img)`
```

```
dat %>%
  ggplot(aes(actual, symptomatic)) +
  geom_point(size = 3, shape = 16) +
  ylim(0, 100) +
  xlim(0, 100) +
  geom_abline(slope = 1, intercept = 0) +
  labs(x = "Quant",
```

```
      y = "pliman")+
  theme_r4pde()
```

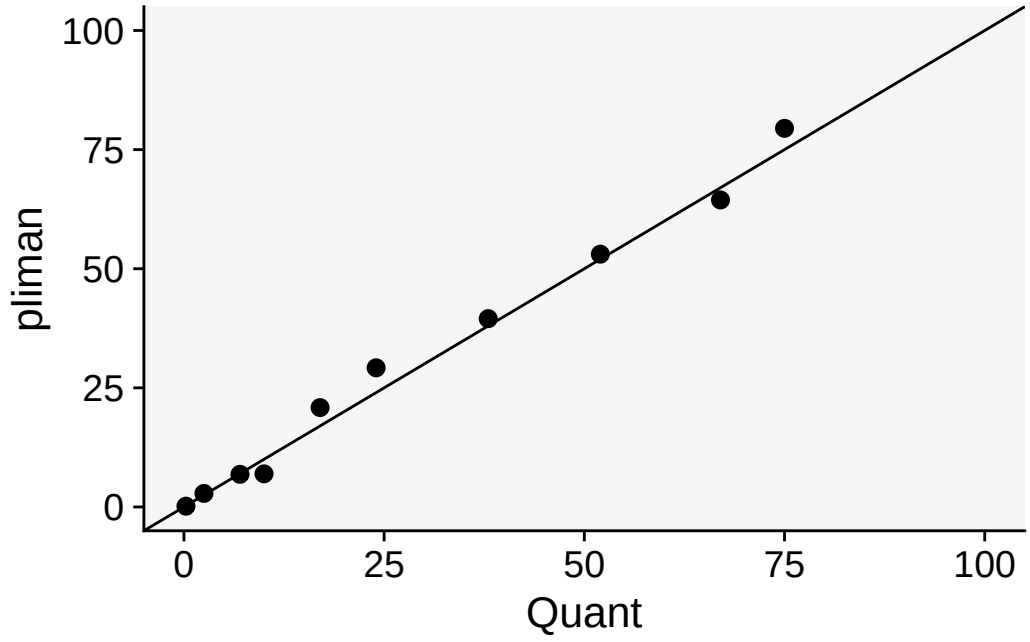


Figure 4.10: Scatter plot for the relationship between severity values measured by pliman and Quant software

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 0.9940835 0.9774587 0.9984566
```

In conclusion, as mentioned earlier, the most critical step is defining the reference image palettes. A few preliminary runs may be necessary for some images to ensure that the segmentation is being carried out correctly, based on visual judgment. This is not different from any other color-threshold based methods, where the choices made by the user impact the final

result and contribute to variation among assessors. The drawbacks are the same as those encountered with direct competitors, namely, the need for images to be taken under uniform and controlled conditions, especially with a contrasting background.

4.5 Creating palettes interactively

Pliman offers another function `measure_disease_iter()` which allows the user to pick up samples in the image to create the color palettes for each required class (background, healthy and symptoms). Check the video below.

https://www.youtube.com/embed/fI_Mm-GlPyw

5 Reliability and accuracy

6 Severity data

6.1 Terminology

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.

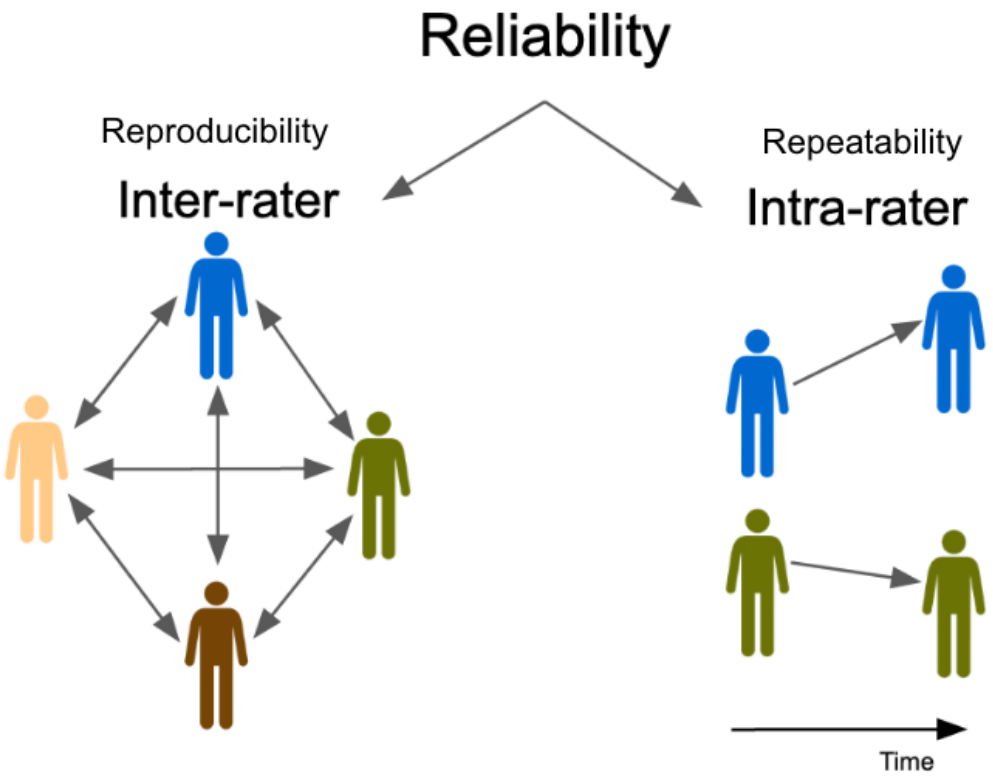


Figure 6.1: Two types of reliability of estimates or measures of plant disease intensity

```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 = "#99cc66")+
  geom_point(aes(x = 5, y = 5), size = 60, color = "white")+
  geom_point(aes(x = 5, y = 5), size = 50, color = "#99cc66")+
  geom_point(aes(x = 5, y = 5), size = 40, color = "white")+
  geom_point(aes(x = 5, y = 5), size = 30, color = "#99cc66")+
  geom_point(aes(x = 5, y = 5), size = 20, color = "white")+
  geom_point(aes(x = 5, y = 5), size = 10, color = "#99cc66")+
  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)
```

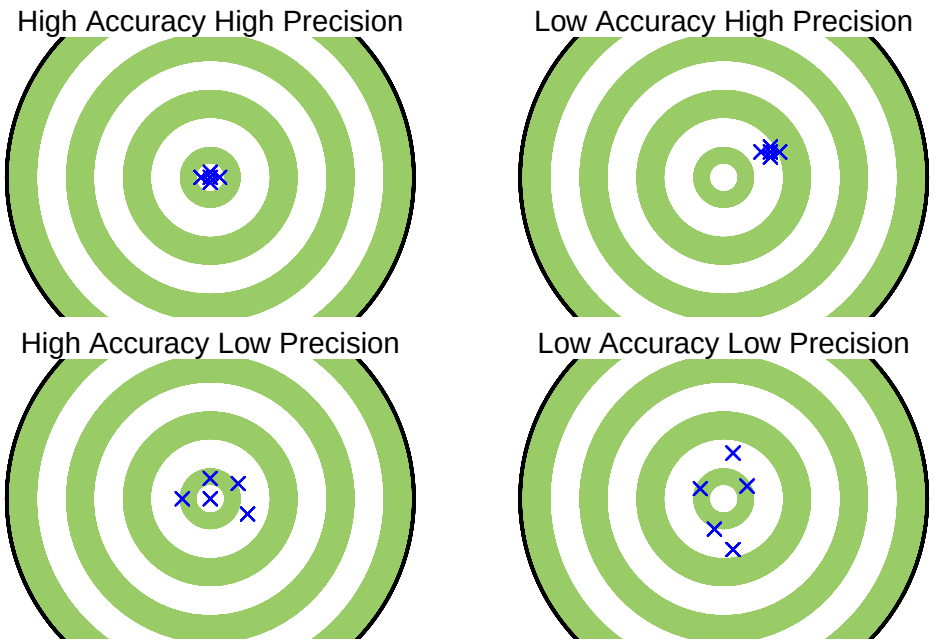


Figure 6.2: The accuracy and precision of the archer is determined by the location of the group of arrows

Another way to visualize accuracy and precision is via scatter plots for the relationship between the actual values and the estimates.

```r
library(tidyverse)
library(r4pde)
theme_set(theme_r4pde())
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 = "#99cc66", size = 3)+
   ylim(0,70)+
  xlim(0,70)+
  labs(x = "Actual", y = "Estimate",
       subtitle = "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 = "#99cc66", size = 3)+
  ylim(0,70)+
  xlim(0,70)+
  labs(x = "Actual", y = "Estimate",
       subtitle = "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 = "#99cc66", size = 3)+
  ylim(0,70)+
  xlim(0,70)+
  labs(x = "Actual", y = "Estimate",
       subtitle = "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 = "#99cc66", size = 3)+
  ylim(0,70)+
  xlim(0,70)+
  labs(x = "Actual", y = "Estimate",
       subtitle = "Low Acccuracy Low Precision")

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

6.2 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.

6.2.1 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.

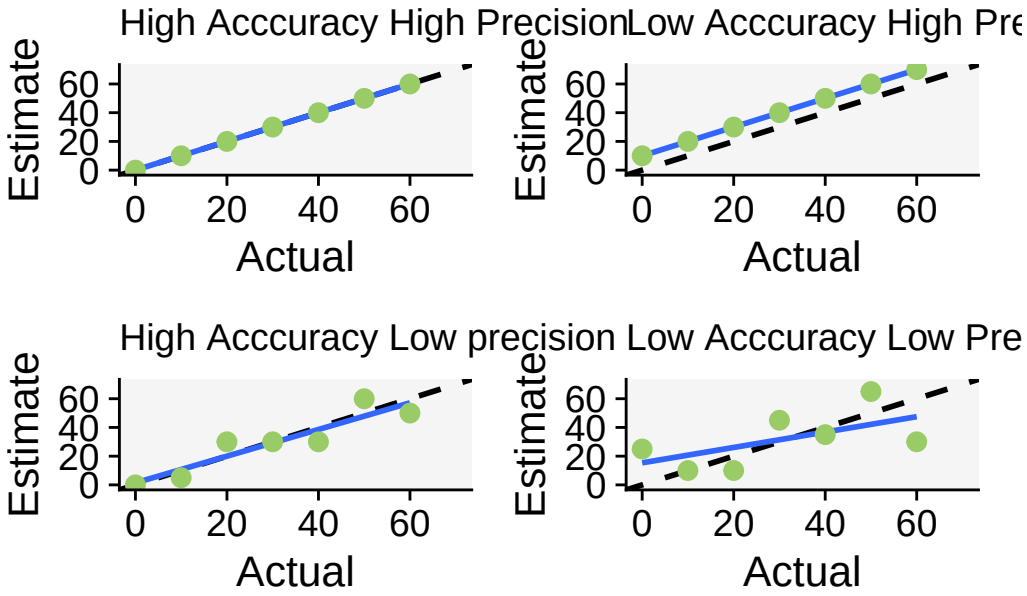


Figure 6.3: Scatter plots for the relationship between actual and estimated values representing situations of low or high precision and accuracy. The dashed line indicates the perfect concordance and the solid blue line represents the fit of the linear regression model

```
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.

```
# 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)+
  labs(y = "Severity estimate (%)",
       x = "Leaf number",
       color = "Rater")
```

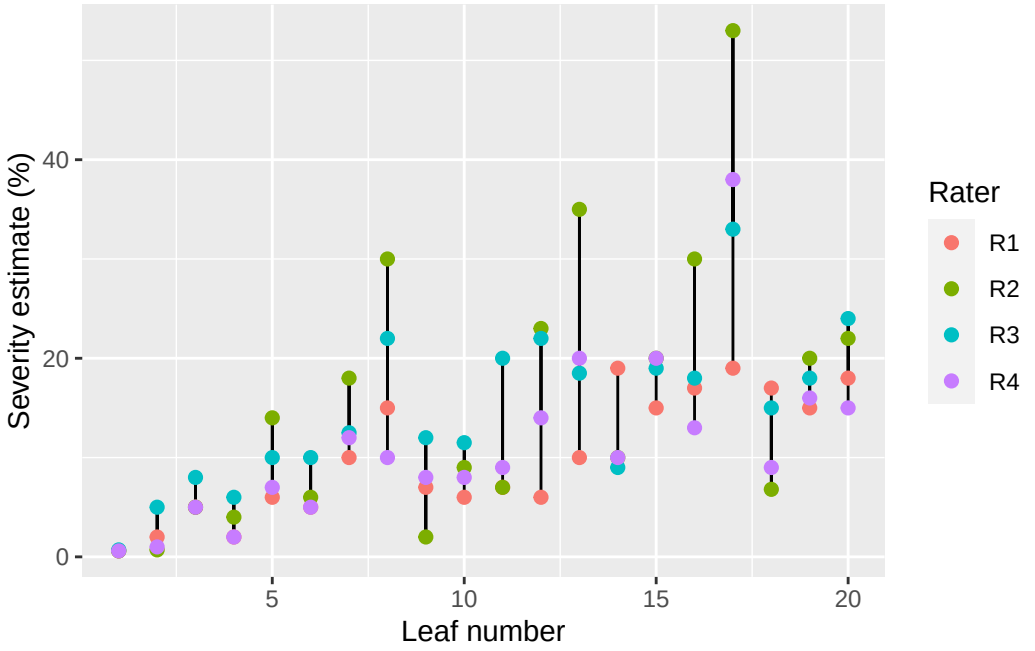


Figure 6.4: Visual estimates of soybean rust severity for each leaf by each of four raters

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)


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

ggpairs(raters)+
  theme_r4pde()
```

6.2.1.1 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 `cor` function,

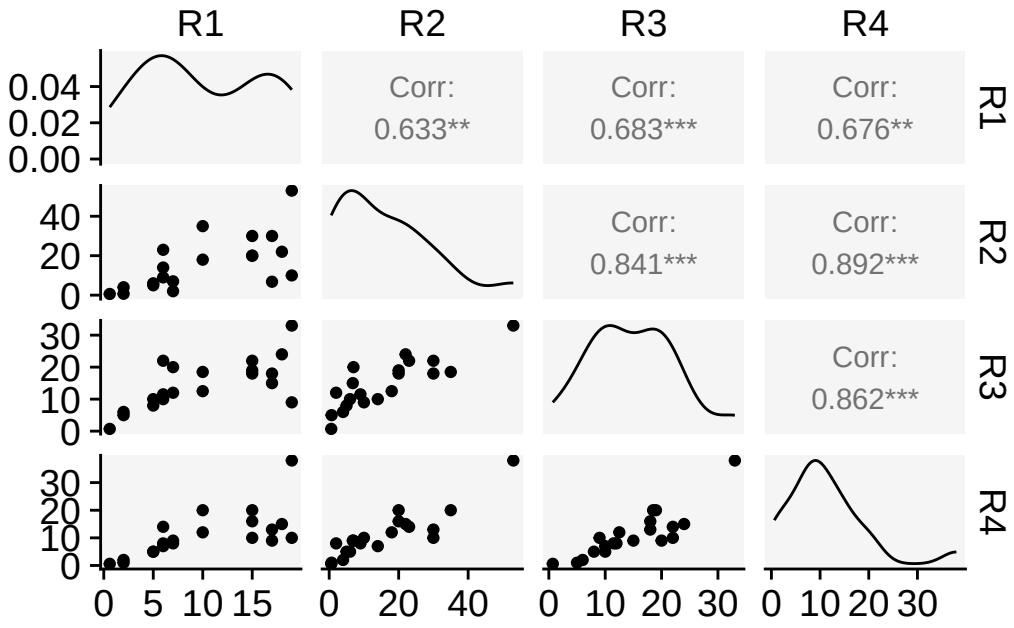


Figure 6.5: Correlation plots relating severity estimates for all pairs of raters

```
knitr::kable(cor(raters))
```

Table 6.1: Pearson correlation coefficients for all pairs of raters

	R1	R2	R3	R4
R1	1.0000000	0.6325037	0.6825936	0.6756986
R2	0.6325037	1.0000000	0.8413333	0.8922181
R3	0.6825936	0.8413333	1.0000000	0.8615470
R4	0.6756986	0.8922181	0.8615470	1.0000000

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

```
# All pairwise R2

raters_cor <- reshape2::melt(cor(raters))

raters2 <- raters_cor |>
  filter(value != 1)
```

```r
# means of R2
raters2$value
```

```
 [1] 0.6325037 0.6825936 0.6756986 0.6325037 0.8413333 0.8922181 0.6825936
 [8] 0.8413333 0.8615470 0.6756986 0.8922181 0.8615470
```

```r
round(mean(raters2$value^2), 3)
```

```
[1] 0.595
```

6.2.1.2 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.

```r
library(psych)
ic <- ICC(raters)
knitr::kable(ic$results[1:2]) # only selected columns
```

	type	ICC
Single_raters_absolute	ICC1	0.6405024
Single_random_raters	ICC2	0.6464122

	type	ICC
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
```

6.2.1.3 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
```

6.2.2 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.

6.2.3 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 6.6). 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)+
  theme_r4pde()+
  labs(x = "Actual severity (%)",
       y = "Estimate severity (%)")
```

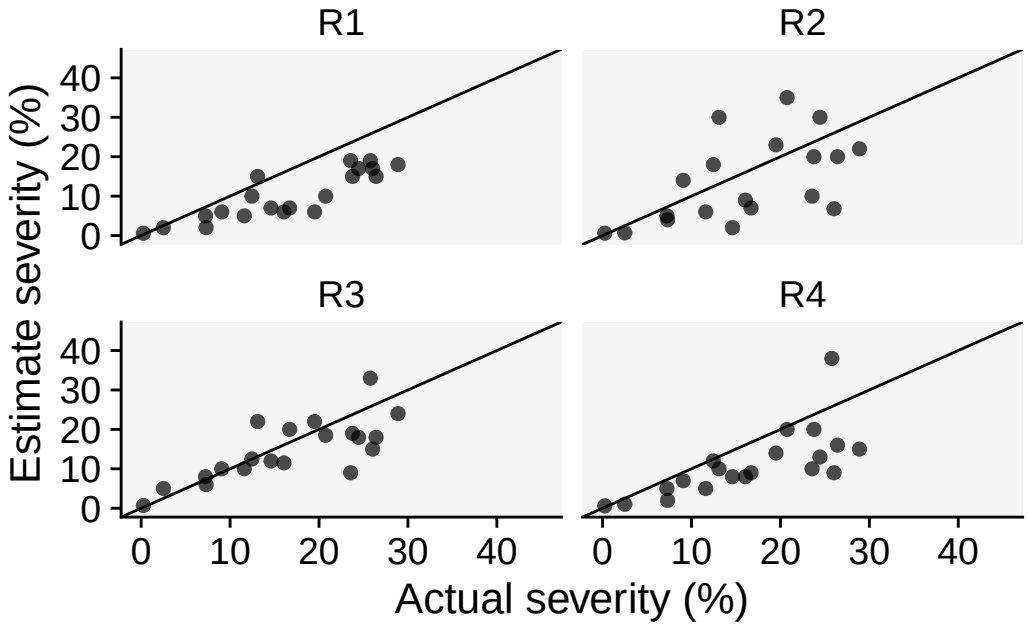


Figure 6.6: Scatterplots for the relationship between estimated and actual severity for each rater

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::correlation()

knitr::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
```

6.2.4 Accuracy

6.2.4.1 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)+
  theme_r4pde()+
  labs(x = "Actual severity (%)",
       y = "Error (Estimate - Actual)")
```

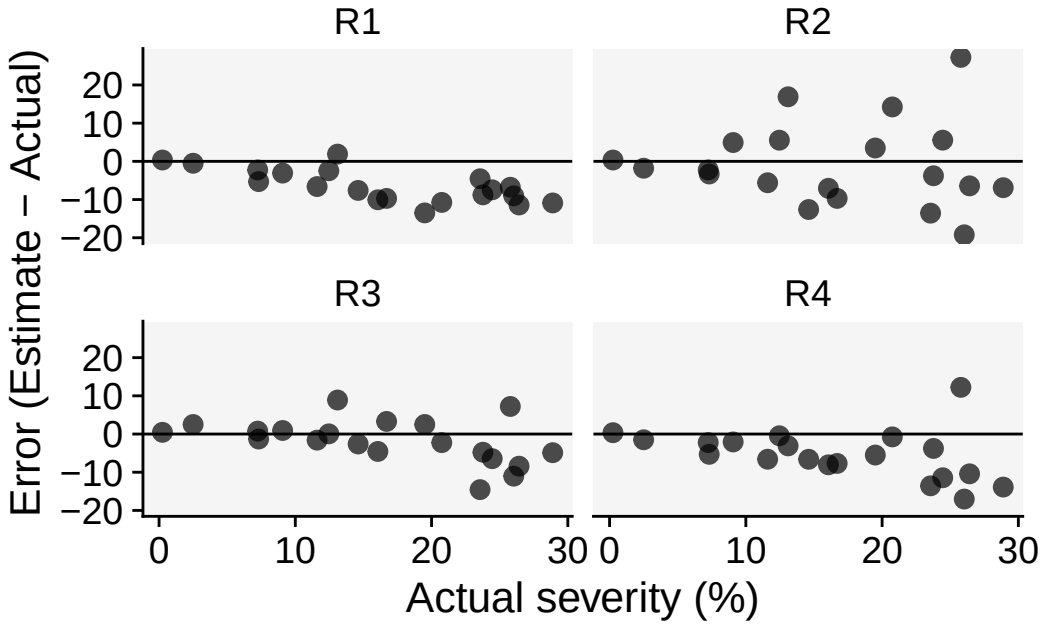


Figure 6.7: Error (estimated - actual) of visual severity estimates

6.2.4.2 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
# 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)
knitr::kable(res)
```

Table 6.4: Statitics of the concordance correlation coefficient summarizing accuracy and precision of visual severity estimates of soybean rust for a single rater

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

Now let's create a function that will allow us to estimate the CCC for all raters in the data frame in the wide format. The function assumes that the first two columns are the actual and estimates and the rest of the columns are the raters, which is the case for our `sbr` dataframe . Let's name this function `ccc_byrater`.

```
ccc_byrater <- function(data) {
  long_data <- pivot_longer(data, cols = -c(leaf, actual),
                            names_to = "rater", values_to = "measurement")
  ccc_results <- long_data %>%
    group_by(rater) %>%
    summarise(Agreement = as.numeric(epi.ccc(measurement, actual)$rho.c[1]),
              `Bias coefficient` = epi.ccc(measurement, actual)$C.b,
              Precision = Agreement * `Bias coefficient`,
              scale_shift = epi.ccc(measurement, actual)$s.shift,
              location_shift = epi.ccc(measurement, actual)$l.shift)

  return(ccc_results)
}
```

Then, we use the `ccc_byrater` function with the original `sbr` dataset - or any other dataset in the wide format of similar structure. The output is a dataframe with all CCC statistics.

```
results <- ccc_byrater(sbr)
knitr::kable(results)
```

rater	Agreement	Bias coefficient	Precision	scale_shift	location_shift
R1	0.5968136	0.6839766	0.4082065	1.3652694	0.9090386
R2	0.5230656	0.8948494	0.4680649	0.6214585	0.0666069
R3	0.7306948	0.9701226	0.7088635	1.1028813	0.2280303
R4	0.5861371	0.8245860	0.4833205	1.0044929	0.6522573

7 Incidence data

7.1 Accuracy

Incidence data are binary at the individual level; an individual is diseased or not. Here, different from severity that is estimated, the specimen is classified. Let's create two series of binary data, each being a hypothetical scenario of assignment of 12 plant specimens into two classes: healthy (0) or diseased (1).

```
order <- c(1:12)
actual <- c(1,1,1,1,1,1,1,1,0,0,0,0)
class <- c(0,0,1,1,1,1,1,1,1,0,0,0)

dat_inc <- data.frame(order, actual, class)
dat_inc
```

```
   order actual class
1      1      1     0
2      2      1     0
3      3      1     1
4      4      1     1
5      5      1     1
6      6      1     1
7      7      1     1
8      8      1     1
9      9      0     1
10    10      0     0
11    11      0     0
12    12      0     0
```

In the example above, the rater makes 9 accurate classification and misses 3: 2 diseased plants classified as being disease-free (sample 1 and 2), and 1 healthy plant that is wrongly classified as diseased (sample 9).

Notice that there are four outcomes:

TP = true positive, a positive sample correctly classified
TN = true negative, a negative sample correctly classified
FP = false positive, a negative sample classified as positive
FN = false negative, a positive sample classified as positive.

There are several metrics that can be calculated with the help of a confusion matrix, also known as error matrix. Considering the above outcomes, here is a how a confusion matrix looks like.

Suppose a 2x2 table with notation

	Actual value	
Classification value	Diseased	Healthy
Diseased	TP	FP
Healthy	FN	TN

Let's create this matrix using a function of the caret package.

```
library(caret)
attach(dat_inc)
cm <- confusionMatrix(factor(class), reference = factor(actual))
cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction 0 1
         0 3 2
         1 1 6

               Accuracy : 0.75
                 95% CI : (0.4281, 0.9451)
    No Information Rate : 0.6667
    P-Value [Acc > NIR] : 0.3931

                  Kappa : 0.4706

 Mcnemar's Test P-Value : 1.0000

            Sensitivity : 0.7500
            Specificity : 0.7500
         Pos Pred Value : 0.6000
```

```
         Neg Pred Value : 0.8571
             Prevalence : 0.3333
         Detection Rate : 0.2500
   Detection Prevalence : 0.4167
      Balanced Accuracy : 0.7500

       'Positive' Class : 0
```

The function returns the confusion matrix and several statistics such as accuracy = (TP + TN) / (TP + TN + FP + FN). Let's manually calculate the accuracy and compare the results:

```
TP = 3
FP = 2
FN = 1
TN = 6
accuracy = (TP+TN)/(TP+TN+FP+FN)
accuracy
```

```
[1] 0.75
```

Two other important metrics are sensitivity and specificity.

```
sensitivity = TP/(TP+FN)
sensitivity
```

```
[1] 0.75
```

```
specificity = TN/(FP+TN)
specificity
```

```
[1] 0.75
```

We can calculate some metrics using the *MixtureMissing* package.

```
library(MixtureMissing)
evaluation_metrics(actual, class)
```

```
$matr
       pred_0 pred_1
true_0      3      1
true_1      2      6

$TN
[1] 3

$FP
[1] 1

$FN
[1] 2

$TP
[1] 6

$TPR
[1] 0.75

$FPR
[1] 0.25

$TNR
[1] 0.75

$FNR
[1] 0.25

$precision
[1] 0.8571429

$accuracy
[1] 0.75

$error_rate
[1] 0.25

$FDR
[1] 0.1428571
```

7.2 Reliability

```
library(psych)
tab <- table(class, actual)
phi(tab)
```

```
[1] 0.48
```

8 Standard area diagrams

8.1 Definitions

According to a glossary on phytopathometry (Bock et al. 2021b), standard area diagram (SAD) can be defined as "*a generic term for a pictorial or graphic representation (drawing or true-color photo) of selected disease severities on plants or plant parts (leaves, fruit, flowers, etc.) generally used as an aid for more accurate visual estimation (on the percentage scale) or classification (using an ordinal scale) of severity on a specimen".*

The Standard Area Diagrams (SADs), also known as diagrammatic scales, have a long history of use in plant pathology. The concept dates back to the late 1800s when the Cobb scale was developed, featuring five diagrams depicting a range of severity levels of rust pustules on wheat leaves.

In the past 20 years, plant pathologists have leveraged advancements in image processing and analysis tools, along with insights from psychophysical and measurement sciences, to develop SADs that are realistic (e.g., true-color photographs), validated, and depict severities that maximize estimation accuracy. SADs have been created in various color formats (black or white, two-color, or true-color) and with varying incremental scales (approximated linear or logarithmic) (Del Ponte et al. 2017).

SADs have proven beneficial in increasing the accuracy of visual estimates, as estimating percentage areas is generally more challenging than classifying severity into ordinal classes - there are numerous possibilities on the percentage scale, compared to the finite and small number of classes in ordinal scales. A recent quantitative review confirmed that using SADs often results in improved accuracy and precision of visual estimates. However, it also identified factors related to SAD design and structure, disease symptoms, and actual severity that affected the outcomes. In particular, SADs have shown greater utility for raters who are inherently less accurate and for diseases characterized by small and numerous lesions (Del Ponte et al. 2022). Here are examples of SADs in black and white, two-color, and true-color formats:

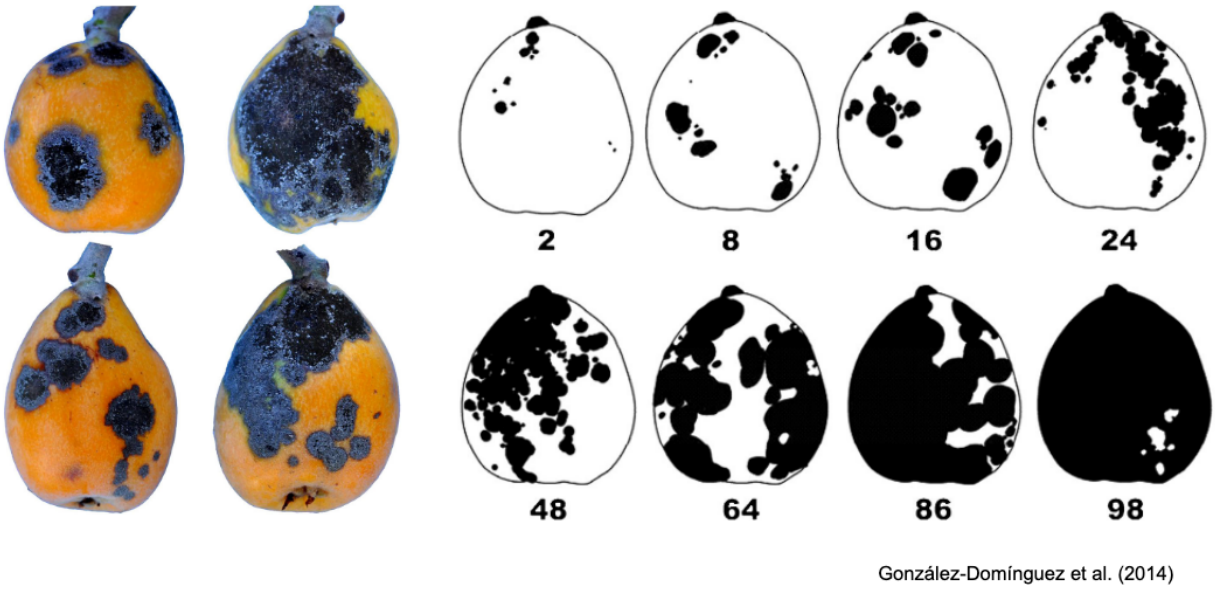


Figure 8.1: Actual photos of symptoms of loquat scab on fruit (left) and a SADs with eight diagrams (right). Each number represents severity as the percent area affected (González-Domínguez et al. 2014)

More SADs can be found in the SADBank, a curated collection of articles on SAD development and validation. Click on the image below to get access to the database.

8.2 SAD development and validation

A systematic review of the literature on SADs highlighted the most important aspects related with the development and validation of the tool (Del Ponte et al. 2017). A list of best practices was proposed in the review to guide future research in the area. Follows the most important aspects to be noted:

Best practices on SADs development

- Sample a minimum number (e.g., n = 100) of specimens from natural epidemics representing the range of disease severity and typical symptoms observed.
- Use reliable image analysis software to discriminate disease symptoms from healthy areas to calculate percent area affected.

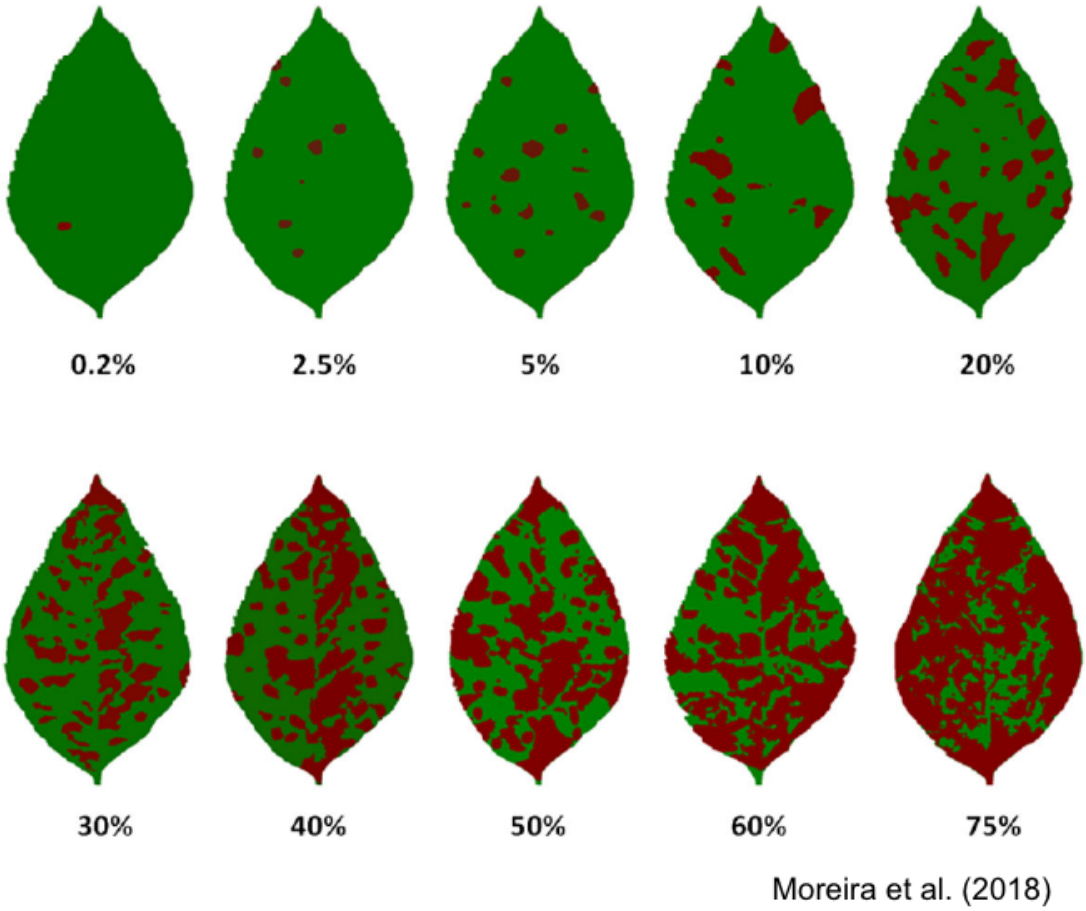


Figure 8.2: SADs for Glomerella leaf spot on apple leaf. Each number represents severity as the percent area affected (Moreira et al. 2018)

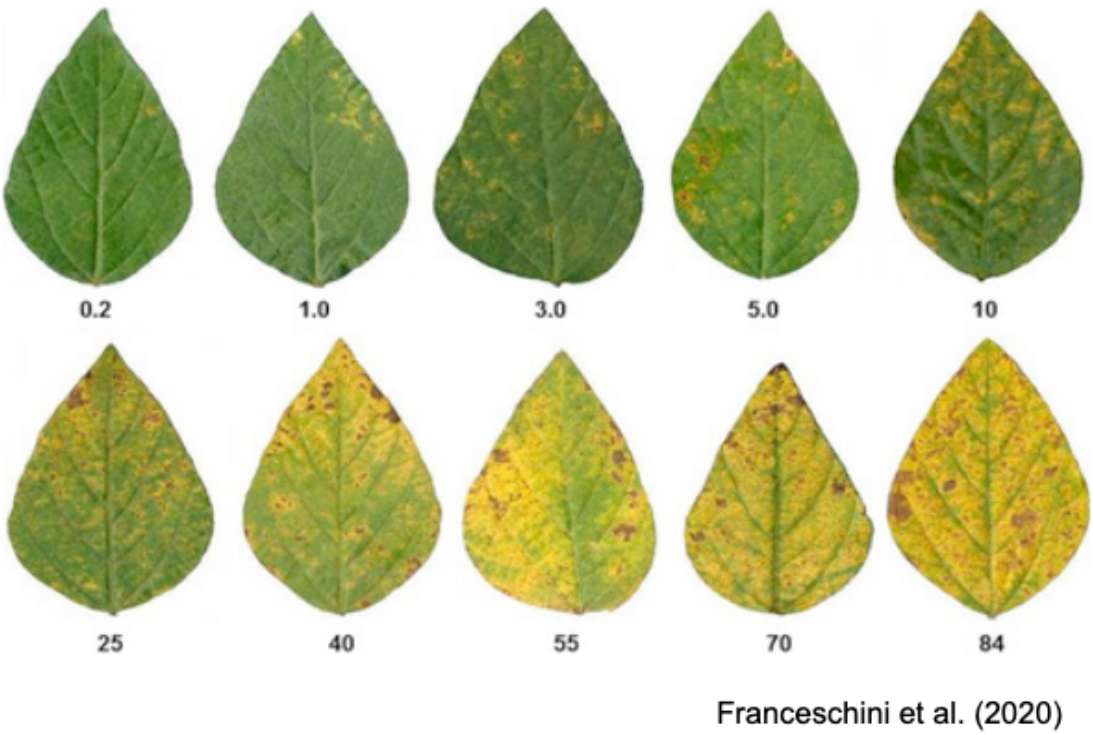


Figure 8.3: SADs for soybean rust. Each number represents severity as the percent area affected (Franceschi et al. 2020)

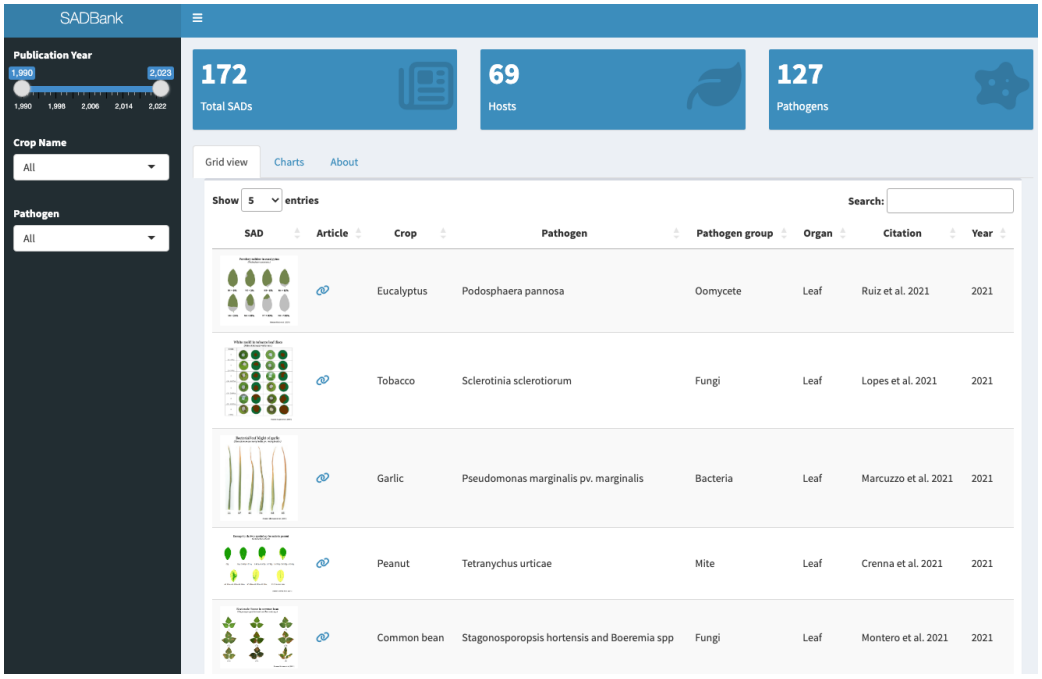


Figure 8.4: SADBank, a curated collection of articles

- When designing the illustrations for the SAD set, ensure that the individual diagrams are prepared realistically, whether line drawn, actual photos, or computer generated.
- The number of diagrams should be no less than 6 and no more than 10, distributed approximately linearly, and spaced no more than 15% apart. Additional diagrams (± 2) should be included between 0 and 10% severity.
- For the validation trial, select at least 50 specimens representing the full range of actual severity and symptom patterns.
- When selecting raters (a minimum of 15) for validation, make sure they do not have previous experience in using the SAD under evaluation.
- Provide standard instructions on how to recognize the symptoms of the disease and how to assess severity, first without and then with the SAD.
- Ideally repeat the assessment in time, with a 1- or 2-week interval, both without and with the aid, using the same set of raters in order to evaluate the effect of training and experience on gains in accuracy.

- Both pre- and posttest experiment conditions should be the same to avoid any impact of distraction on accuracy of estimates during the tests.

8.3 Designing SADs in R

The diagrams used in a set have been developed using various methods and technologies, ranging from hand-drawn diagrams to actual photographs (Del Ponte et al. 2017). There is an increasing trend towards using actual photos that are digitally analyzed using standard image analysis software to determine the percent area affected. With this approach, a large set of images is analyzed, and some images are chosen to represent the severities in the SAD according to the scale structure.

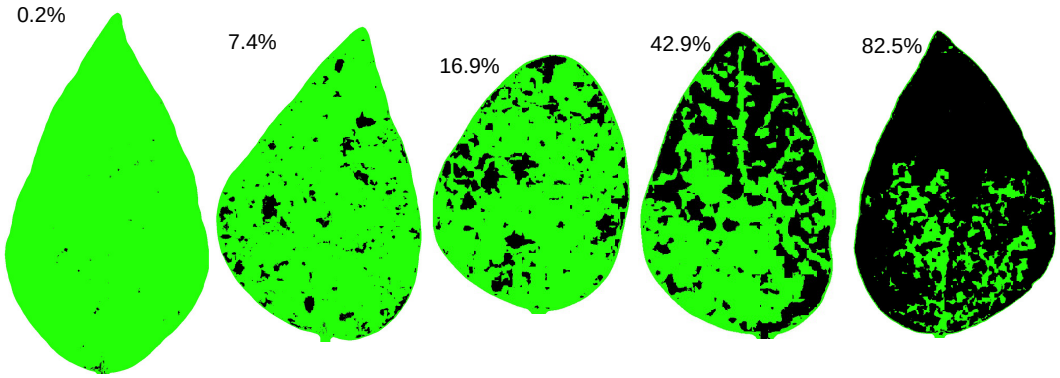
In R, the pliman package has a function called `sad()` which allows the automatic generation of a SADs with a pre-defined number of diagrams. Firstly, as shown in the previous chapter, the set of images to be selected needs to be analysed using the `measure_disease()` function. Then, a SADs is automatically generated. In the function, the specimens with the smallest and highest severity will be selected for the SAD. The intermediate diagrams are sampled sequentially to achieve the pre-defined number of images after the severity has been ordered from low to high. More details of the function here.

Let's use the same set of 10 soybean leaves, as seen in the previous chapter, depicting the rust symptoms and create the `sbr` object.

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

sbr <- 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_original = FALSE, # set to TRUE for showing the original.
  col_background = "white",
  verbose = FALSE,
  plot = FALSE
)
```

We are ready to run the `sad()` function to create a SADs with five diagrams side by side. The resulting SADs is in two-color as standard. Set the argument `show_original` to `TRUE` for showing the orignal image in the SADs.

```
sad(sbr, 5, ncol = 5)
```



8.4 Analysis of SADs validation data

To evaluate the effect of SAD on accuracy components, analyze the data, preferably using concordance analysis methods (see chapter), to fully explore which component is affected and to gain insight into the ramification of errors. Linear regression should not be used as the sole method but it could be complementary for comparison with previous literature.

Inferential methods should be used for testing hypotheses related to gain in accuracy. If parametric tests are used (paired t-test for example), make sure to check that the assumptions are not violated. Alternatively, nonparametric tests (Wilcoxon signed rank) or nonparametric bootstrapping should be used when the conditions for parametric tests are not met. More recently, a (parametric) mixed modelling framework has been used to analyse SADs validation data where raters are taken as a random effects in the model (González-Domínguez et al. 2014; Franceschi et al. 2020; Pereira et al. 2020).

8.4.1 Non parametric boostrapping of differences

Bootstrap is a resampling method where large numbers of samples of the same size are repeatedly drawn, ***with replacement***, from a single original sample. It is commonly used when the distribution of a statistic is unknown or complicated and the sample size is too small to draw a valid inference.

A bootstrap-based equivalence test procedure was first proposed as complementary to parametric (paired t-test) or non-parametric (Wilcoxon) to analyze severity estimation data in a study on the development and validation of a SADs for pecan scab (Yadav et al. 2012). The equivalence test was used to calculate 95% confidence intervals for each statistic by bootstrapping using the percentile method (with an equivalence test, the null hypothesis is the converse of H0, i.e. the null hypothesis is non-equivalence). In that study, the test was used to compare means of the CCC statistics across raters under two conditions: 1) without versus with the SAD; and 2) experienced versus inexperienced raters.

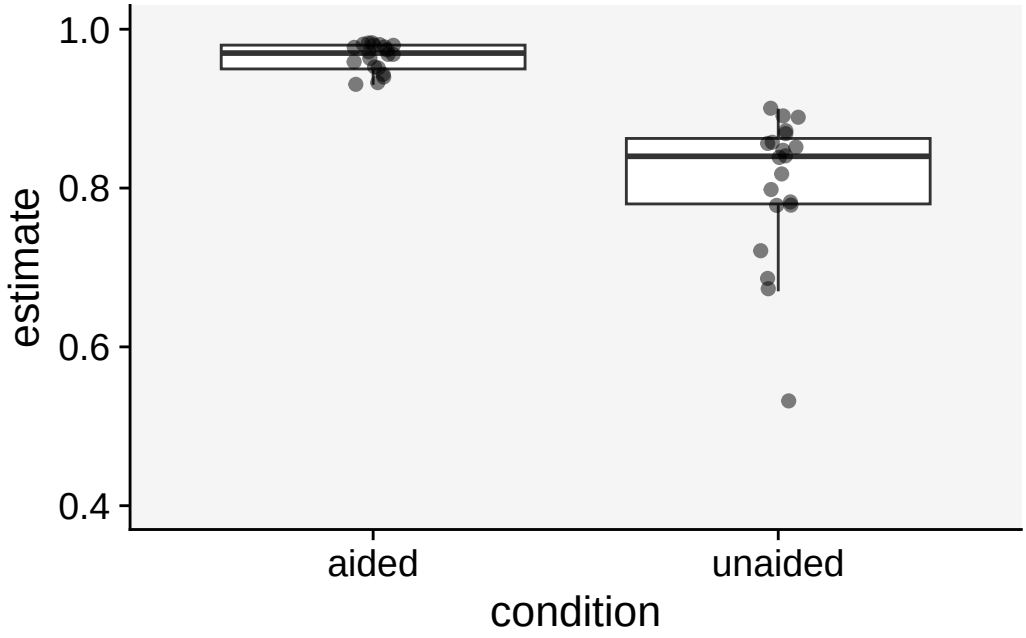
To apply the bootstrap-based equivalence test, let's work with the CCC data for a sample of 20 raters who estimated severity of soybean rust SAD first without and then with the aid. The CCC was calculated as shown here.

```
library(tidyverse)
library(r4pde)

sbr <- tibble::tribble(
  ~rater, ~aided, ~unaided,
      1,   0.97,     0.85,
      2,   0.97,     0.85,
      3,   0.95,     0.82,
      4,   0.93,     0.69,
      5,   0.97,     0.84,
      6,   0.96,     0.86,
      7,   0.98,     0.78,
      8,   0.93,     0.72,
      9,   0.94,     0.67,
     10,   0.95,     0.53,
     11,   0.94,     0.78,
     12,   0.98,     0.89,
     13,   0.96,      0.8,
     14,   0.98,     0.87,
     15,   0.98,      0.9,
     16,   0.98,     0.87,
     17,   0.98,     0.84,
     18,   0.97,     0.86,
```

```
    19,   0.98,     0.89,
    20,   0.98,     0.78
  )
```

Let's visualize the data using boxplots. Each point in the plot represents a rater.

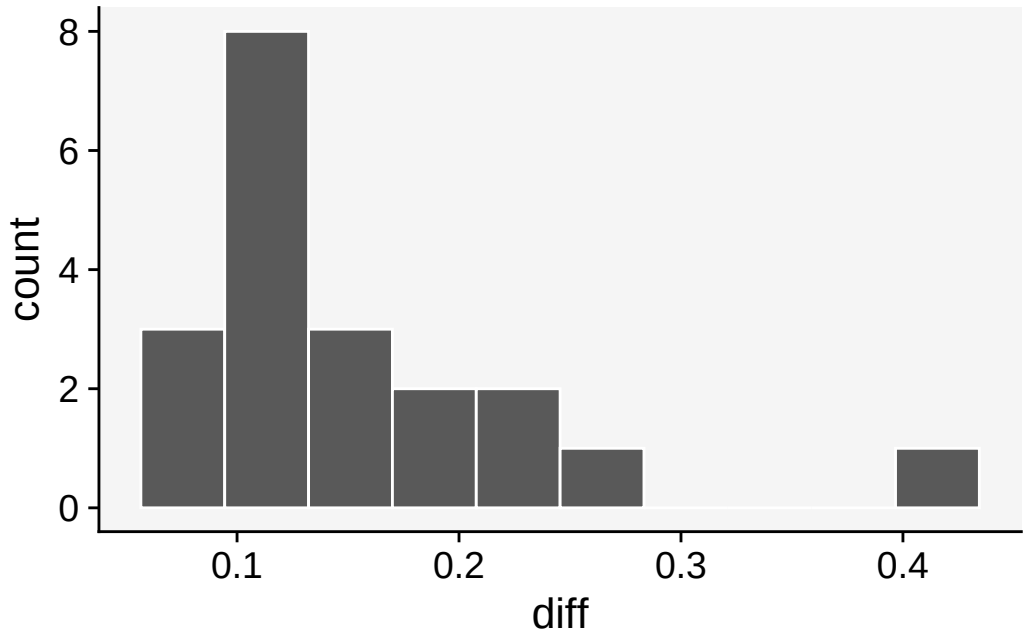
```
theme_set(theme_r4pde())
sbr |>
  pivot_longer(2:3, names_to = "condition", values_to ="estimate") |>
  ggplot(aes(condition, estimate))+
  geom_boxplot(outlier.colour = NA)+
  geom_jitter(width = 0.05, size = 2, alpha = 0.5)+
  theme_r4pde()+
  ylim(0.4,1)
```



To proceed with bootstrapping, we first create a new variable to hold the differences between the means of the estimates (aided minus unaided). If the 95% CI does not include zero, this means that there was a significant improvement in the statistics.

```
# diff of means
sbr$diff <- sbr$aided - sbr$unaided
```

```
sbr |>
  ggplot(aes(x= diff))+
  theme_r4pde()+
  geom_histogram(bins = 10, color = "white")
```



Using the simpleboot and boot packages of R:

```
library(simpleboot)
```

```
Simple Bootstrap Routines (1.1-7)
```

```
b.mean <- one.boot(sbr$diff, mean, 999)
boot::boot.ci(b.mean)
```

```
Warning in boot::boot.ci(b.mean): bootstrap variances needed for studentized
intervals
```

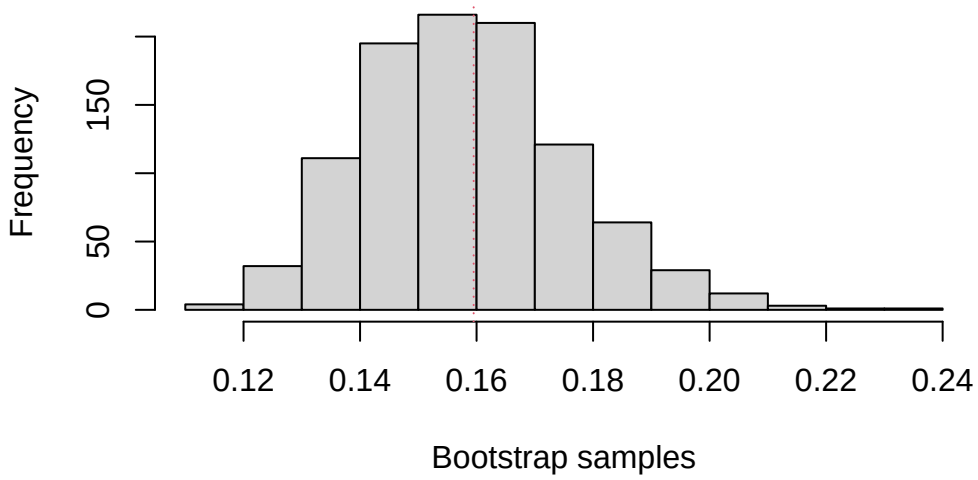
```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates
```

```
CALL :
boot::boot.ci(boot.out = b.mean)

Intervals :
Level      Normal              Basic
95%   ( 0.1261,  0.1946 )   ( 0.1230,  0.1910 )

Level     Percentile            BCa
95%   ( 0.1280,  0.1960 )   ( 0.1295,  0.2000 )
Calculations and Intervals on Original Scale
```

```r
mean(b.mean$data)
```

```
[1] 0.1595
```

```r
hist(b.mean)
```



Using the bootstrap package:

```r
library(bootstrap)
b <- bootstrap(sbr$diff, 999, mean)
```

```
  quantile(b$thetastar, c(.025,.975))
```

```
 2.5% 97.5%
0.128 0.195
```

```
  mean(b$thetastar)
```

```
[1] 0.1577758
```

```
  sd(b$thetastar)
```

```
[1] 0.01731425
```

```
  se <- function(x) sqrt(var(x)/length(x))
  se(b$thetastar)
```

```
[1] 0.0005477987
```

Both procedures shown above have led to similar results. The 95% CIs of the differences did not include zero, so a significant improvement in accuracy can be inferred.

8.4.2 Parametric and non-parametric paired sample tests

When two estimates are gathered from the same rater at different times, these data points are not independent. In such situations, a **paired sample t-test** can be utilized to test if the mean difference between two sets of observations is zero. This test requires each subject (or leaf, in our context) to be measured or estimated twice, resulting in *pairs* of observations. However, if the assumptions of the test (such as normality) are violated, a non-parametric equivalent, such as the Wilcoxon signed-rank test, also known as the **Wilcoxon test**, can be employed. This alternative is particularly useful when the data are not normally distributed.

To proceed with these tests, we first need to ascertain whether our data are normally distributed. We should also verify whether the variances are equal. Let's now apply these two tests to our data and compare the results.

```
  # normality test
  shapiro.test(sbr$aided)
```

```
    Shapiro-Wilk normality test

data:  sbr$aided
W = 0.82529, p-value = 0.002111
```

```r
shapiro.test(sbr$unaided)
```

```
    Shapiro-Wilk normality test

data:  sbr$unaided
W = 0.83769, p-value = 0.003338
```

```r
# equal variance test
var.test(sbr$aided, sbr$unaided)
```

```
    F test to compare two variances

data:  sbr$aided and sbr$unaided
F = 0.037789, num df = 19, denom df = 19, p-value = 1.53e-09
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.01495720 0.09547109
sample estimates:
ratio of variances
        0.03778862
```

```r
# paired t-test
t.test(sbr$aided, sbr$unaided, paired = TRUE)
```

```
    Paired t-test

data:  sbr$aided and sbr$unaided
t = 8.812, df = 19, p-value = 3.873e-08
alternative hypothesis: true difference in means is not equal to 0
```

```
95 percent confidence interval:
 0.1216158 0.1973842
sample estimates:
mean of the differences
                 0.1595
```

```r
# Wilcoxon test
wilcox.test(sbr$aided, sbr$unaided, paired = TRUE)
```

```
Warning in wilcox.test.default(sbr$aided, sbr$unaided, paired = TRUE): cannot
compute exact p-value with ties
```

```
    Wilcoxon signed rank test with continuity correction

data:  sbr$aided and sbr$unaided
V = 210, p-value = 9.449e-05
alternative hypothesis: true location shift is not equal to 0
```

As shown above, the two assumptions were violated, so we could rely more confidently on the non-parametric test.

8.4.3 Mixed effects modeling

Mixed models, also known as mixed effects models or multilevel models, are an extension of traditional linear models that are used for analyzing hierarchical or clustered data. These models are particularly useful when dealing with data where observations may not be fully independent, or when the assumption of independence is violated. This happens, for instance, when data are collected over time from the same individuals or units, or when individuals are grouped or nested within higher-level units, such as in our case where measurements are taken by different raters (Brown 2021).

Mixed models enable us to model both fixed and random effects. Fixed effects represent the usual regression parameters that we are primarily interested in estimating, while random effects model the random variation that occurs at different levels of hierarchy or clustering. They allow us to account for variability among different levels of data, like inter-rater variability or intra-subject variability in repeated measures designs.

In our context, we consider raters as random effects because we view them as a sample drawn from a larger population of potential raters, and our goal is to generalize our findings to this larger population. If we were to sample additional raters, we would expect these new raters to

differ from our current ones. However, by considering raters as a random effect in our model, we can account for this inter-rater variability and make more accurate inferences about the overall population.

The random effects component in the mixed model allows us to capture and model the additional variance that is not explicitly accounted for by the fixed effects in our model. In other words, random effects help us to capture and quantify the 'unexplained' or 'residual' variation that exists within and between the clusters or groups in our data. This could include, for instance, variation in disease measurements that are taken repeatedly from the same subjects. In conclusion, mixed models provide a robust and flexible framework for modeling hierarchical or clustered data, allowing us to effectively account for both fixed and random effects and to make more accurate inferences about our data.

Let's start reshaping our data to the long format and assign them to a new data frame.

```
sbr2 <- sbr |>
  pivot_longer(2:3, names_to = "condition", values_to = "estimate")
```

Now we fit the mixed model using the `lmer` function of the *lme4* package. We will fit the model to the logit of the estimate because they should be bounded between zero and one. Preliminary analysis using non-transformed or log-transformed data resulted in lack of normality of residuals and heterocedasticity (not shown).

```
library(lme4)
library(car) # for logit function
mix <- lmer(logit(estimate) ~ condition + (1 | rater), data = sbr2)

# Check model performance
library(performance)
check_normality(mix)
```

```
OK: residuals appear as normally distributed (p = 0.381).
```

```
check_heteroscedasticity(mix)
```

```
OK: Error variance appears to be homoscedastic (p = 0.961).
```

```
# Check effect of condition
car::Anova(mix)
```

```
Analysis of Deviance Table (Type II Wald chisquare tests)

Response: logit(estimate)
           Chisq Df Pr(>Chisq)
condition 458.44  1  < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

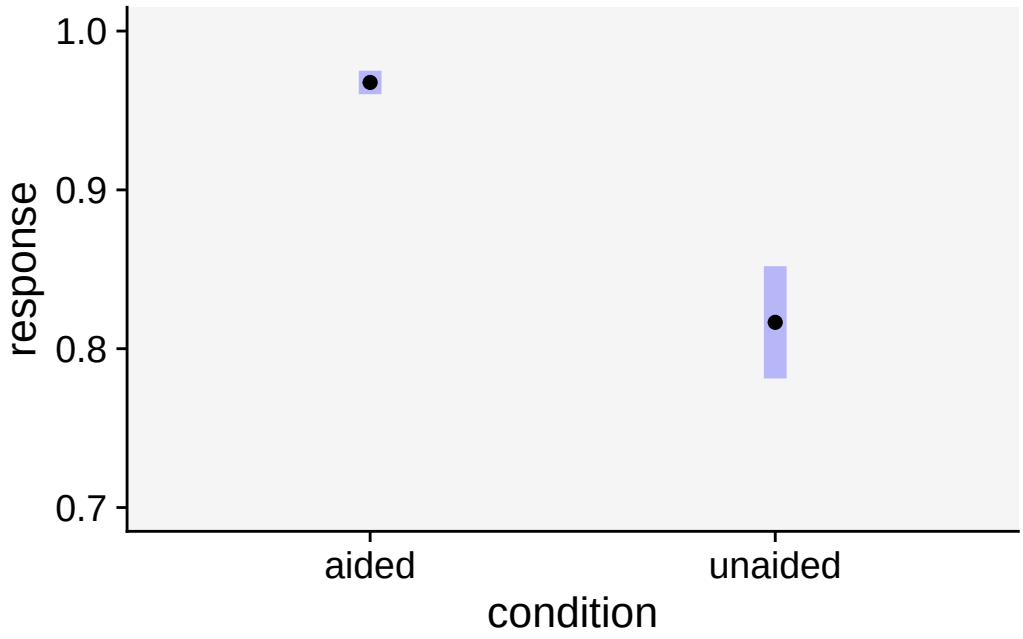
```r
# Estimate the means for each group
library(emmeans)
em <- emmeans(mix, ~ condition, transform = "response")
em
```

```
 condition response      SE   df lower.CL upper.CL
 aided        0.968 0.00359 25.5    0.960    0.975
 unaided      0.817 0.01719 25.5    0.781    0.852

Degrees-of-freedom method: inherited from kenward-roger when re-gridding
Confidence level used: 0.95
```

```r
# Contrast the means
pairs(em)
```

```
 contrast        estimate     SE   df t.ratio p.value
 aided - unaided    0.151 0.0149 25.5  10.141  <.0001

Degrees-of-freedom method: inherited from kenward-roger when re-gridding
```

```r
# plot the means with 95% CIs
plot(em) +
  coord_flip()+
  xlim(0.7,1)+
  theme_r4pde()
```



As shown above, we can reject the null hypothesis that the means are the same between the two groups.

Alternatively, we could fit GLMMs - generalized linear mixed models, which extend the traditional linear mixed models to accommodate response variables that follow different distributions. They are particularly useful when the response variable does not follow a normal distribution and cannot be adequately transformed to meet the parametric assumptions of traditional linear models. The glmmTMB package in R provides a convenient and flexible platform to fit GLMMs using a variety of distributions (Brooks et al. 2017).

In our case, considering our response variable bounded between 0 and 1, a Beta distribution might be a suitable choice. Beta distribution is a continuous probability distribution defined on the interval [0, 1], and is commonly used for modelling variables that represent proportions or percentages.

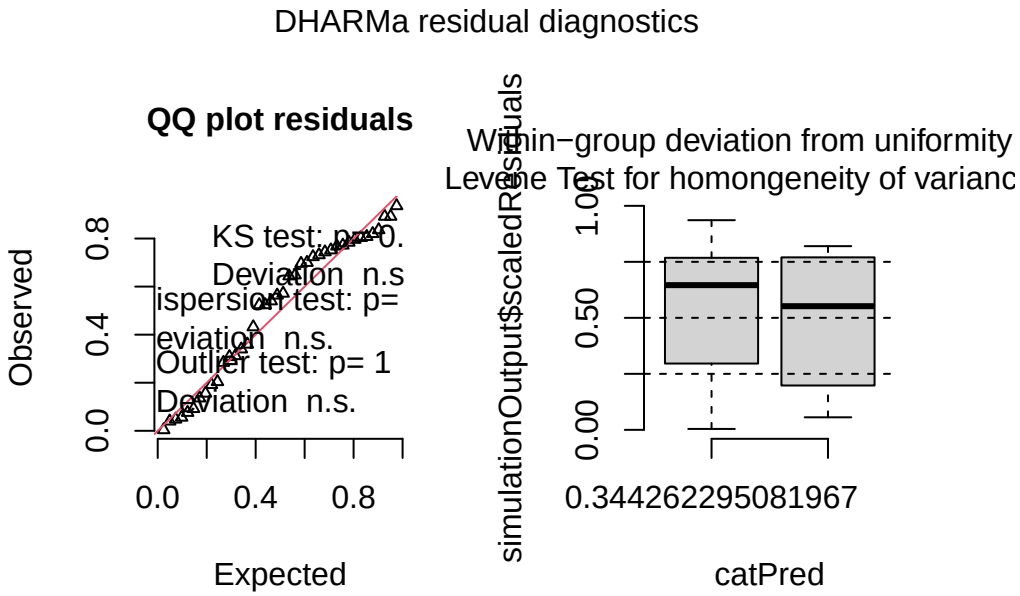
The function `glmmTMB()` from the *glmmTMB* package can be used to fit a GLMM with a Beta distribution. In this function, we specify the distribution family as `beta_family()`.

```
library(glmmTMB)
mix2 <-  glmmTMB(estimate ~ condition + (1| rater),
                 data = sbr2,
                 family = beta_family())
```

Because the package *performance* does not handle the *glmmTMB* output, we will use the *DHARMa* package in R which can be particularly useful for checking the assumptions of

your GLMM fitted with `glmmTMB()`. The package provides a convenient way to carry out residual diagnostics for models fitted via maximum likelihood estimation, including GLMMs. This package creates standardized residuals from the observed responses and the predicted responses of a fitted model, and then compares these residuals to a simulated set of residuals under a correct model.

```
library(DHARMa)

plot(simulateResiduals(mix2))
```



In this example, `simulateResiduals()` generates simulated residuals from your fitted model, and the plot creates a plot of these residuals. This showed that the residuals from our model are uniformly distributed, which is an assumption of GLMMs. We can now proceed with the posthoc analysis and noticed that the results are similar to when the response variable was transformed to logit.

```
car::Anova(mix2)
```

```
Analysis of Deviance Table (Type II Wald chisquare tests)

Response: estimate
            Chisq Df Pr(>Chisq)
condition 400.93  1  < 2.2e-16 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
library(emmeans)
em <- emmeans(mix2, ~ condition, transform = "response")
em
```

```
 condition response     SE df lower.CL upper.CL
 aided        0.967 0.0043 36    0.958    0.976
 unaided      0.814 0.0167 36    0.780    0.848

Confidence level used: 0.95
```

```
# Contrast the means
pairs(em)
```

```
 contrast        estimate     SE df t.ratio p.value
 aided - unaided    0.153 0.0139 36  11.001  <.0001
```

9 Training sessions

9.1 Training sessions

In the same way that Standard Area Diagrams (SADs) can improve the accuracy of visual estimates of disease severity, exposure to a diverse set of diagrams or actual images with known severity values can significantly enhance a rater's assessment proficiency. The key to this improvement is the frequent exposure to various levels of severity, which enables the rater to better calibrate their judgments over time.

As the rater engages with these reference diagrams or images, they develop a mental model of the severity scale. This mental model is continually refined through repeated exposure to a variety of severity values. This iterative learning process allows the rater to adjust their estimations based on the feedback from known values, thus improving their overall accuracy and precision in disease severity estimation.

Such a process, often termed 'training', is particularly beneficial in scenarios where visual estimation is the primary tool for assessing disease severity. Training raters using sets of reference images is an effective strategy to enhance inter-rater reliability and consistency over time, especially when coupled with other tools like SADs.

9.2 Software

Indeed, the use of computerized training sessions in assessing disease severity has a rich history, dating back to the mid-1980s when personal computers were first introduced. These early applications were developed using operating systems like DOS or Windows and involved software like AREAGRAM, DISTRAIN, DISEASE.PRO, ESTIMATE, SEVERITY.PRO, and COMBRO. These programs utilized computerized images with specific and measured disease severities to train raters, as outlined in the review by Bock et al. (2021a).

The main advantage of these computerized training sessions is that they allow raters to familiarize themselves with various disease severity levels, thereby enhancing their performance in severity estimation. Such training has been proven to significantly improve the accuracy and consistency of disease severity evaluations.

However, a potential limitation of this approach is the short-lived nature of the benefits derived from such training. The skills and proficiency gained from these computerized training sessions

may degrade over time, necessitating regular retraining for raters to maintain their performance level. This could be due to the fact that estimation skills, like many other skills, require regular practice for maintenance. Without ongoing exposure to severity scales and continued practice, the accuracy and precision of a rater's estimates may decline.

To address this challenge, it would be beneficial to implement a structured training regimen that includes regular retraining sessions. This could help ensure the continued proficiency of raters in estimating disease severity, thus maintaining the accuracy and reliability of assessments over time. Furthermore, it would be advantageous to investigate the optimal frequency and structure of these training sessions to maximize their effectiveness and sustainability in the long term.

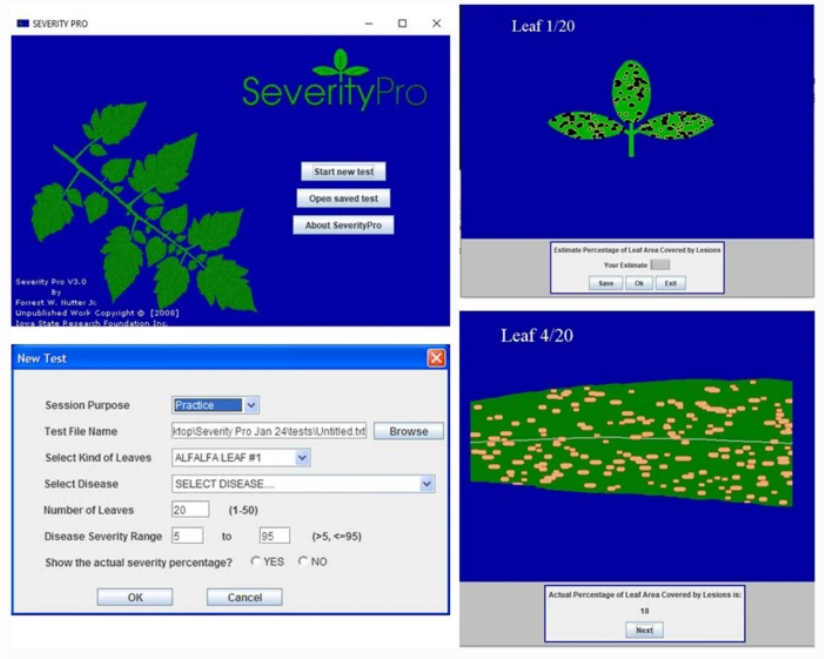


Figure 9.1: Selected screenshots from Severity.Pro, the disease assessment training program by Forrest W. Nutter (Madden et al. 2021).

9.2.1 Online training tools

In Brazil, the "Sistema de treinamento de acuidade visual" was initially developed as a web-based system to train raters in assessing citrus canker. The system has evolved over time and now has a current version that is accessible on both iOS and Android platforms. You can find the current version of the system at this link. This platform provides an interactive training experience to enhance the ability of raters in accurately assessing the severity of citrus canker.

In Mexico, a specific application called **Validar-PER** has been developed to train raters in visually assessing the severity of coffee leaf rust. This application utilizes diagrammatic log-based scales as a standardized approach for severity assessment. You can access the Validar-PER application online here. The application aims to improve the proficiency of raters in evaluating the severity of coffee leaf rust using a systematic and standardized methodology.

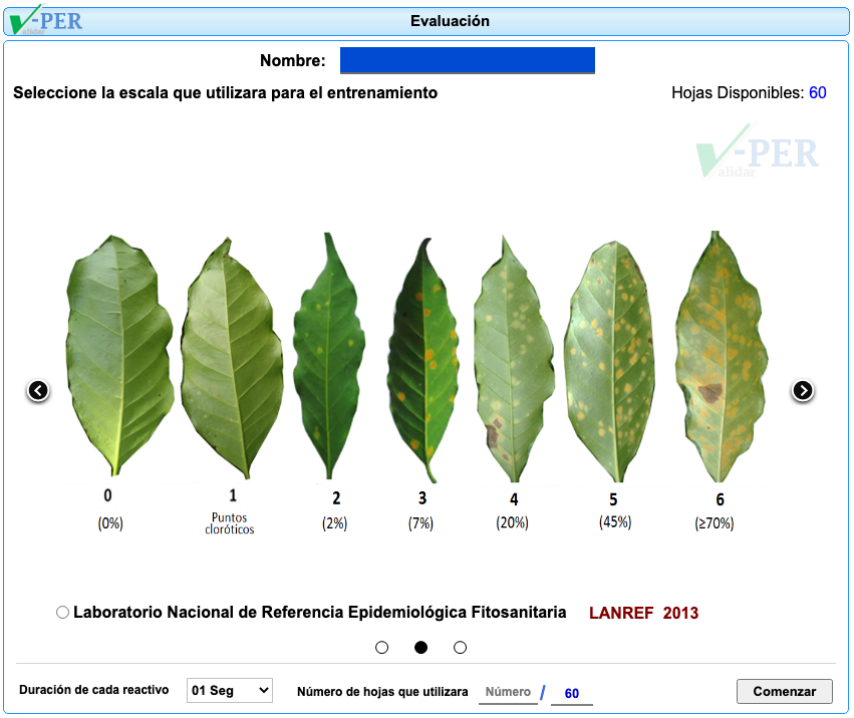


Figure 9.2: Screen of Validar-PER, an online training module for assessing coffee leaf rust severity

9.2.2 Training software made with R

9.2.2.1 TraineR

TraineR, developed by the author of this book, is created using R and Shiny. Its purpose is to train users in assessing disease severity, specifically expressed as the percentage area of an organ (leaf or fruit) affected by lesions.

To use the app, users can adjust parameters for organ shape, organ color, as well as lesion shape, lesion color, lesion number, and lesion size. These adjustments will generate a standard area diagram with an ellipsoidal shape.

To initiate the training, users should first set the desired number of attempts for the session and click on the "generate new" button. A diagram will then be displayed, and users should input their estimate of the diseased area as a numeric value in percentage. The estimate will be recorded and shown in a table along with the actual value, enabling a comparison between the actual and estimated values.

Users can continue generating new diagrams and providing estimates until they reach the defined number of attempts. Once the final attempt is completed, the app will present the accuracy in the form of Lin's concordance correlation coefficient to the user. Plots depicting the relationship between estimates and actual values, as well as the error of the estimates, will be displayed. Furthermore, comprehensive accuracy statistics are also made available.

Currently, the app has certain limitations, including the inability to overlap lesions and a maximum severity representation of approximately 60%. Nonetheless, it remains a valuable educational and demonstration tool.

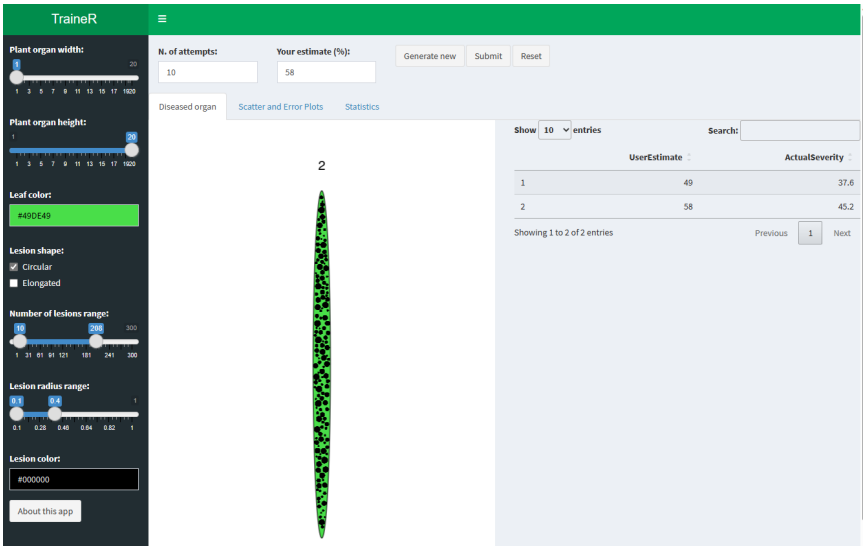


Figure 9.3: Screen of TraineR, an online app for training in the assessment of plant disease severity

9.2.2.2 Trainer2

Trainer2 the second generation of TraineR, takes advantage of actual photographs showcasing disease symptoms. This updated version allows for testing the ability of raters to assess disease severity, particularly by evaluating the percentage area affected based on real symptoms captured in the photographs.

By utilizing actual images, Trainer2 offers a more realistic and practical approach to training raters. Raters can now evaluate disease severity by visually inspecting the symptoms depicted

in the photographs, enhancing their ability to accurately assess the extent of damage in terms of the affected area.

The incorporation of real symptoms in Trainer2 serves as a valuable tool for evaluating and refining the skills of raters in disease severity assessment. It provides a more authentic training experience and helps raters become proficient in identifying and quantifying the extent of disease based on visual cues observed in real-life scenarios.

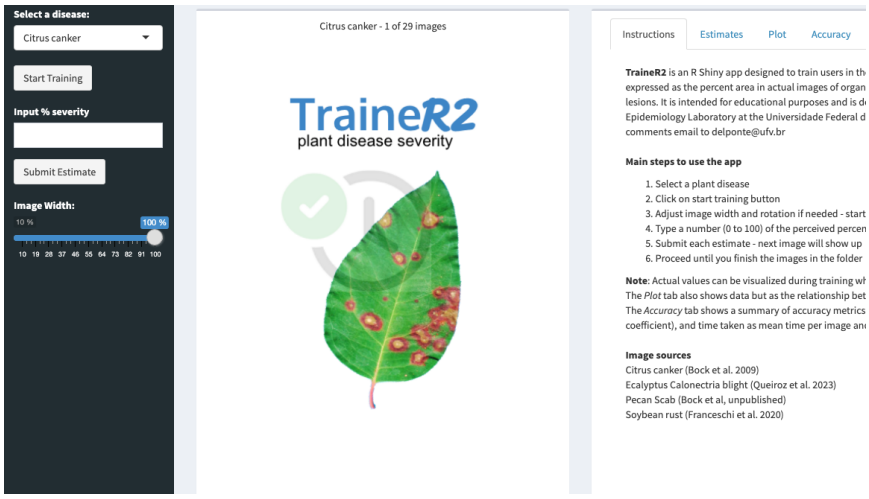


Figure 9.4: Screen of traineR2, an online for training in the assessment of plant disease severity based on real symptoms captured in photographs

Part II

Temporal analysis

10 Disease progress curves

10.1 How epidemics occur

Before knowing how epidemics develop in time, it is important to understand how an epidemic occur. An epidemic begins when the **primary inoculum** (a variable number of propagules able to infect the plant) that is *surviving* somewhere establishes an intimate contact with individuals of the host population - this process is called *infection*. These inocula are usually surviving externally to the plant host and need to *disperse* (move), passively or by means of a vector, to reach the plant. It can also be that a growing host encounter a localized (static) source of inoculum.

Once the infection is established, the pathogen *colonizes* the plant tissues and disease symptoms are noticed. When this happens, the **incubation period** can be measured in time units. A successful colonization will lead to *reproduction* of the pathogen inside and/or external to the crop, and so the **latent period** is completed, and can also be measure in time units. Finally, the **infectious period** takes place and continues until the pathogen is not capable of producing the **secondary inoculum** on the infected site.

Epidemiologists are generally interested in determining the length of the incubation, latent, and infectious periods as influenced by factors related to the host, pathogen, or environment. This is relevant because the longer it takes for the completion of the incubation and latent periods, the lower the potential number of repeated cycles. In summary, a single "infection cycle" represents all events that occur from infection to dispersal, and this occurs only once for many diseases, while for others there may be multiple cycles, which are defined as an "infection chain."

```
library(tidyverse)
library(r4pde)
periods <- tibble::tribble(
  ~period, ~length, ~color, ~order,
  "Incubation", 10, 0, 1,
  "Latent" , 15, 0, 2,
  "Infectious", 25, 15, 3
)
```

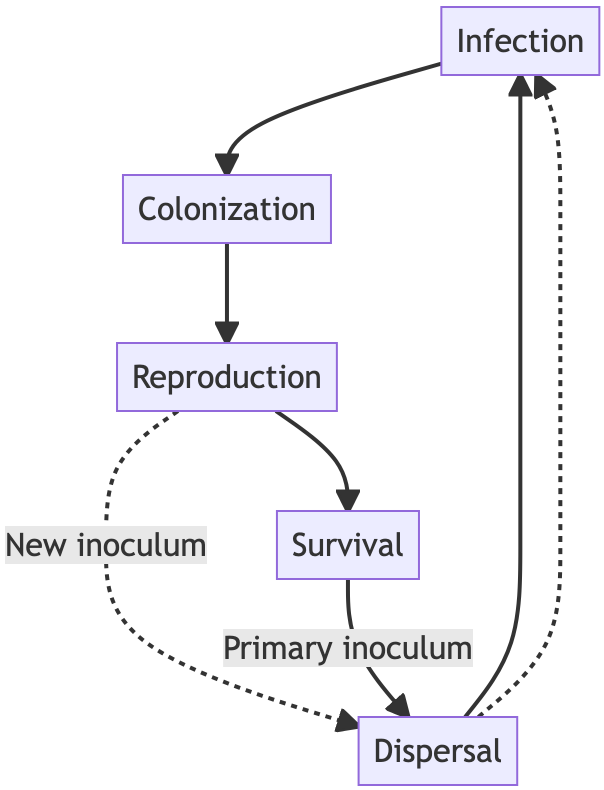


Figure 10.1: Five main processes of the disease cycle

```
p <- periods |>
  ggplot(aes(reorder(period, order), length, fill = period))+
  geom_col()+
  geom_col(aes(period, color), color = "white", fill = "white")+
  coord_flip()+
  theme_void()+
  theme(legend.position = "none")+
  annotate(geom = "text", x = 0.5, y = 15, label = "----- Time --->")+
  annotate(geom = "text", x = 1, y = 5, label = "Incubation", color = "white")+
  annotate(geom = "text", x = 2, y = 8, label = "Latent", color = "white")+
  annotate(geom = "text", x = 3, y = 20, label = "Infectious", color = "white")+
  annotate(geom = "text", x = 1, y = 10.5, label = "Visible symptoms", angle = 90, size =
  annotate(geom = "text", x = 2, y = 15.5, label = "Reproduction starts", angle = 90, size
  annotate(geom = "text", x = 3, y = 25.5, label = "Reproduction ends", angle = 90, size =
  scale_fill_manual(values = c("darkgreen",  "brown", "darkorange"))+
  geom_segment(mapping=aes(x=0.6, y=0, xend=0.6, yend=10), arrow=arrow(ends='both'), size=
  geom_segment(mapping=aes(x=1.6, y=0, xend=1.6, yend=15), arrow=arrow(ends='both'), size=
   geom_segment(mapping=aes(x=2.6, y=15, xend=2.6, yend=25), arrow=arrow(ends='both'), siz
  library(png)
  library(cowplot)
  incubation <- readPNG("imgs/incubation3.png", native = TRUE)
  latent <- readPNG("imgs/latent3.png", native = TRUE)
  p2 <- p + draw_image(incubation , x = 0.5, y = 13, scale = 5)+
    draw_image(latent , x = 1.5, y = 20, scale = 5)
  ggsave("imgs/periods.png", width =6, height =2, bg = "white")
```

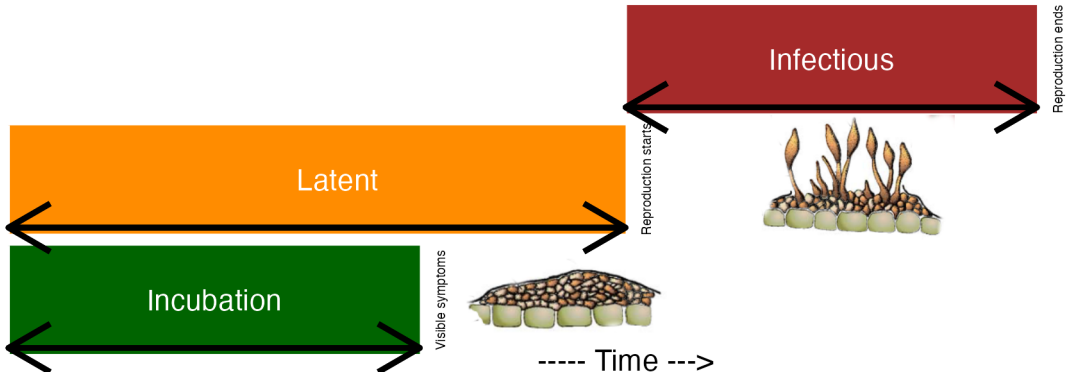


Figure 10.2: Three time-related epidemiological periods and their relations with stages of the disease cycle including colonization (symptoms) and reproduction (sporulation in the case of fungi). Drawings of apple scab symptoms and signs adapted from Agrios (2005)

10.2 Disease curves

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 data frame 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
theme_set(theme_r4pde()) # set global theme
```

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

```
dpc <-
  tribble(
   ~t,  ~y,
   0,  0.1,
   7,  1,
  14,  9,
  21,  25,
  28,  80,
  35, 98,
  42, 99,
  49, 99.9
  )
```

Now the plot

```
dpc1 <- dpc |>
  ggplot(aes(t, y)) +
  theme_r4pde()+
  geom_line(size = 1)+
  geom_point(size = 3, shape = 16)+
```

```
  labs(x = "Assessment time (days)",
       y = "Disease intensity (%)")

ggsave("imgs/dpc1.png", dpc1)
```

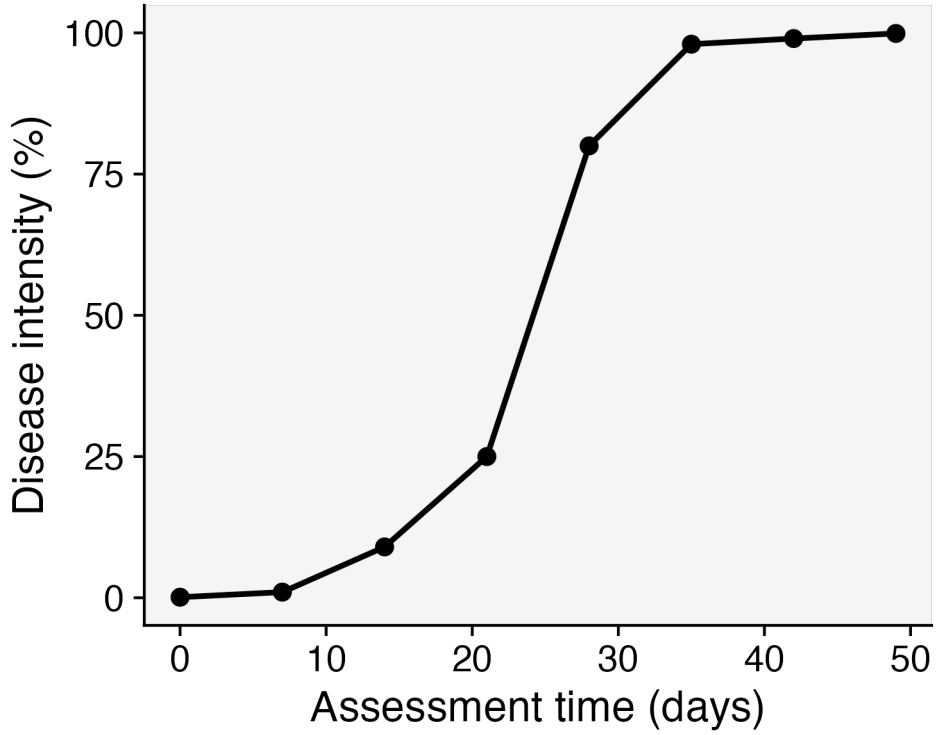


Figure 10.3: A typical disease progress curve for an epidemic that reaches the maximum value

10.3 Epidemic classification

Vanderplank analysed the shapes of great number of epidemic curves and classified the epidemics into two basic types: monocyclic or polycyclic (Vanderplank 1963). In **monocyclic** epidemics, inoculum capable of infecting the crop is not produced during the epidemics. These epidemics are initiated and maintained only by the primary inoculum. There is no secondary infection and hence no further spread of newly produced inoculum among the host individuals. Tipically, the progress curves for monocyclic epidemics have a saturation type shape.

Conversely, when the secondary inoculum produced during the epidemics is capable of infecting the host during the same crop cycle, a **polycyclic** epidemic is established. The number of

repeated cycles just depends on how long it takes to complete a single infection cycle. These epidemics most commonly present a sigmoid shape Figure 10.4.

```
library(tidyverse)
theme_set(theme_bw(base_size = 16))

library(epifitter)
polyc <- sim_logistic(N = 50, dt = 5,
                      y0 = 0.01, r = 0.2,
                      K = 0.8, n = 1,
                      alpha =0)

p <- polyc |>
  ggplot(aes(time, y))+
  geom_point(aes(time, y), size =19, shape =1)+
  geom_line()+
  ylim(0,1)+
  theme_r4pde()+
  labs(x = "Time", y = "Disease intensity")


monoc <- sim_monomolecular(N = 50, dt = 5,
                           y0 = 0.01, r = 0.1,
                           K = 0.8, n = 1,
                           alpha =0)
library(ggforce)
m <- monoc |>
  ggplot(aes(time, y))+
  geom_point(aes(x = 25, y = 0.5), size =90, shape = 1)+
   geom_line()+
 theme_r4pde()+
  ylim(0,1)+
  labs(x = "Time", y = "Disease intensity")

library(patchwork)
cycles <- m | p
ggsave("imgs/cycles.png", bg = "white", width = 8, height =4)
```

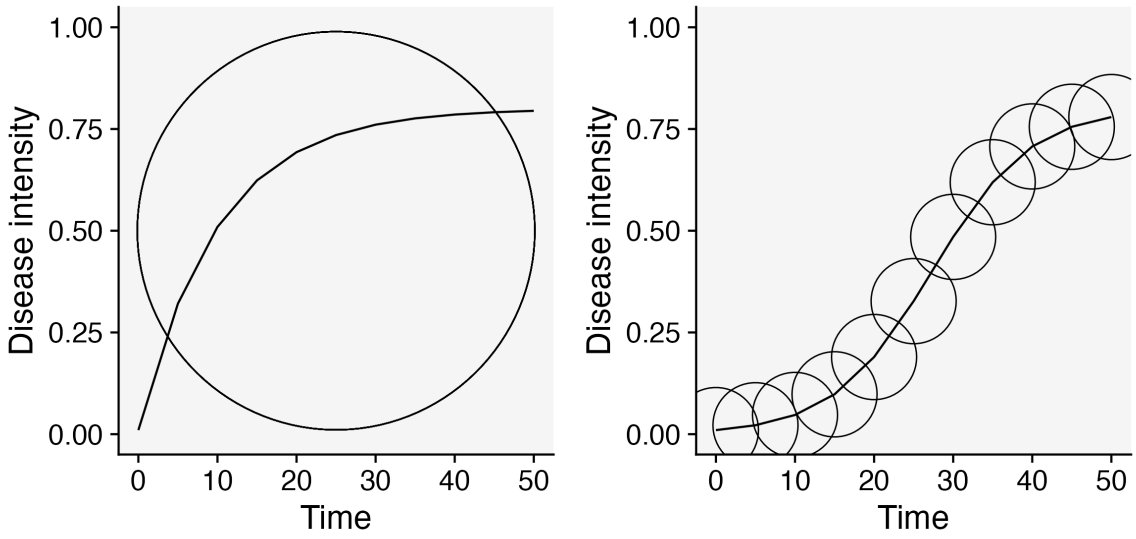


Figure 10.4: Hypothetical curves for monocyclic (left) and polycyclic (right) epidemics. Each circle represents a single infection cycle.

10.4 Curve descriptors and 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 differences 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 quantitative and qualitative descriptors can be extracted including:

- Epidemic duration
- Maximum disease
- Curve shape
- Area under the area under the disease progress curve (AUDPC).

Let's visualize the AUDPC in the same plot that we produced above.

```
dpc2 <- dpc |>
  ggplot(aes(t, y)) +
  labs(x = "Assessment time (days)",
       y = "Disease intensity (%)")+
    geom_area(fill = "#339966")+
    geom_line(linewidth = 1)+
  theme_r4pde()+
  geom_point(size = 3, shape = 16)+
  scale_x_continuous(breaks = c(0, 7, 14, 21, 28, 35, 42))
ggsave("imgs/dpc2.png")
```

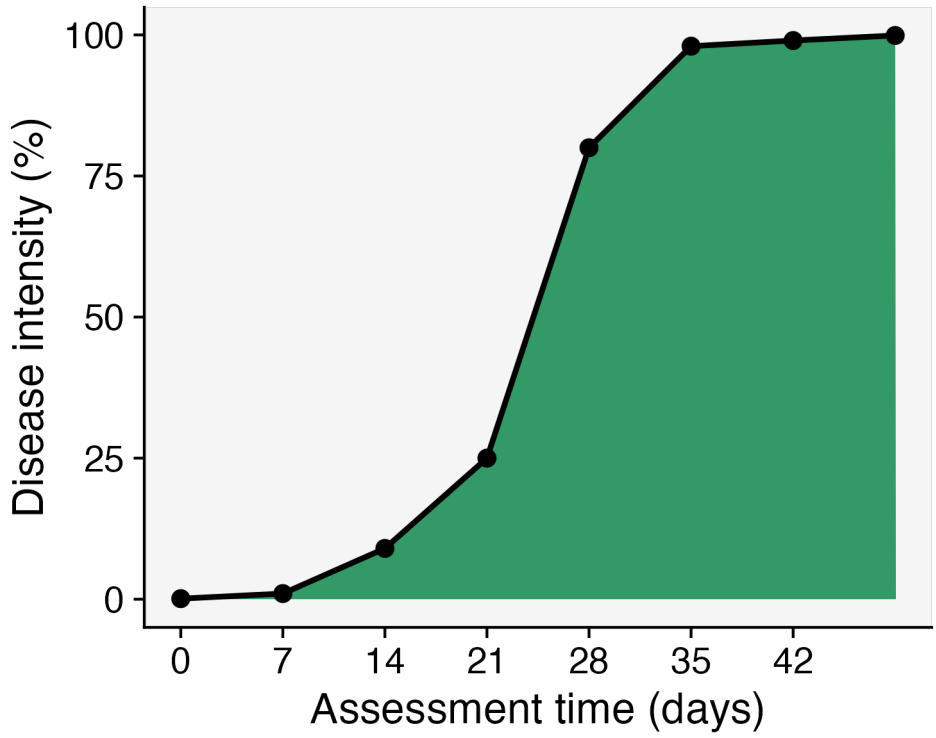


Figure 10.5: Representation of the area under the disease progress curve

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.

```
dpc3 <- dpc |>
  ggplot(aes(t, y)) +
  theme_r4pde()+
  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 = "white", fill = "#339966")+
   annotate("rect", xmin = dpc$t[2], xmax = dpc$t[3],
            ymin = 0, ymax = (dpc$y[2]+ dpc$y[3])/2,
            color = "white", fill = "#339966")+
   annotate("rect", xmin = dpc$t[3], xmax = dpc$t[4],
            ymin = 0, ymax = (dpc$y[3]+ dpc$y[4])/2,
            color = "white", fill = "#339966")+
   annotate("rect", xmin = dpc$t[4], xmax = dpc$t[5],
            ymin = 0, ymax = (dpc$y[4]+ dpc$y[5])/2,
            color = "white", fill = "#339966")+
   annotate("rect", xmin = dpc$t[5], xmax = dpc$t[6],
            ymin = 0, ymax = (dpc$y[5]+ dpc$y[6])/2,
            color = "white",fill = "#339966")+
   annotate("rect", xmin = dpc$t[6], xmax = dpc$t[7],
            ymin = 0, ymax = (dpc$y[6]+ dpc$y[7])/2,
            color = "white", fill = "#339966")+
  annotate("rect", xmin = dpc$t[7], xmax = dpc$t[8],
            ymin = 0, ymax = (dpc$y[7]+ dpc$y[8])/2,
            color = "white", fill = "#339966")+
  geom_line(linewidth = 1)+
  geom_point(size = 3, shape = 16)+
  annotate(geom = "text", x = 36.5, y = 50,
           label = "AUDPC = 2534" , size = 4)+
  scale_x_continuous(breaks = c(0, 7, 14, 21, 28, 35, 42, 49))
ggsave("imgs/dpc3.png")
```

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).

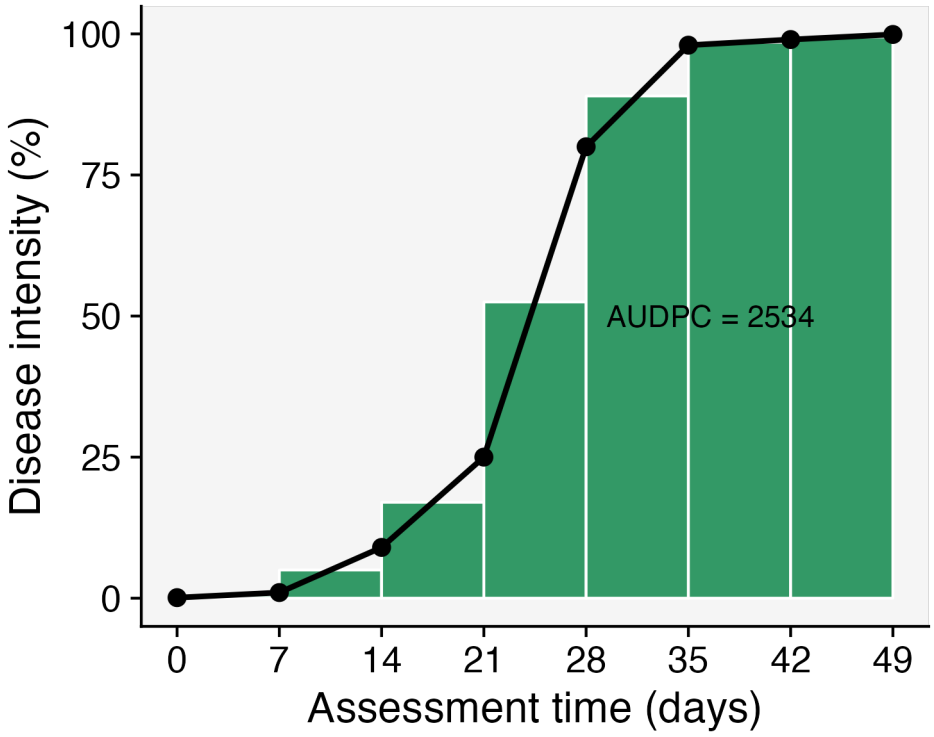


Figure 10.6: Representation of the area under the disease progress curve calculated using the trapezoidal method

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

```
[1] 2534
```

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

```
[1] 0.5171429
```

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

```
[1] 2884
```

11 Population models

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. 2007c). 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.

11.1 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. 2007c) 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.

11.1.1 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)
library(r4pde)
theme_set(theme_r4pde()) # 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) +
  theme_r4pde()+
  labs(x = "Time")
```

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

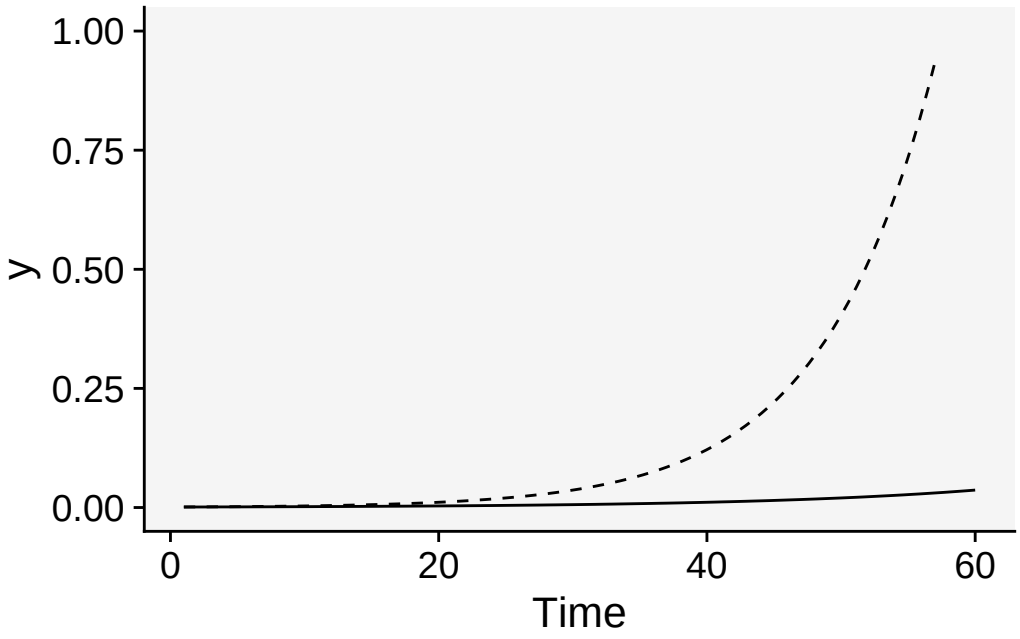


Figure 11.1: Exponential curves with two rates of infection (0.06 and 0.12) and the same initial inoculum (0.001)

```
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) +
  theme_r4pde()+
  labs(x = "Time")
```

11.1.2 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.

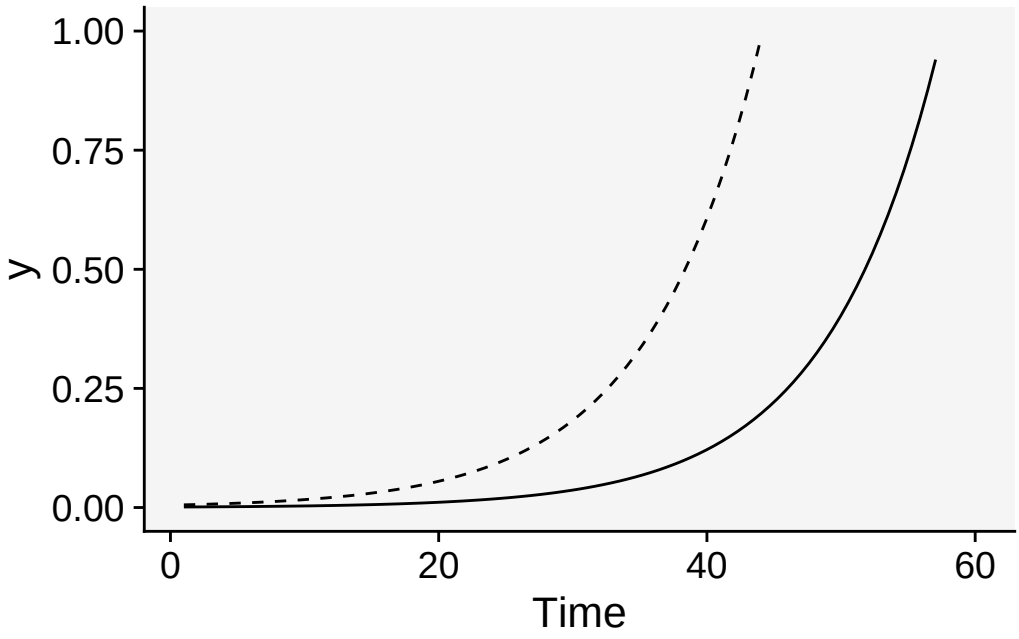


Figure 11.2: Exponential curves with the same rate of infection (0.12) and single and five times the initial inoculum (0.001)

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(x = "Time") +
 theme_r4pde()+
  annotate(geom = "text", x = 35, y = 0.77, label = "r = 0.06") +
  annotate(geom = "text", x = 50, y = 0.55, label = "r = 0.02")
```

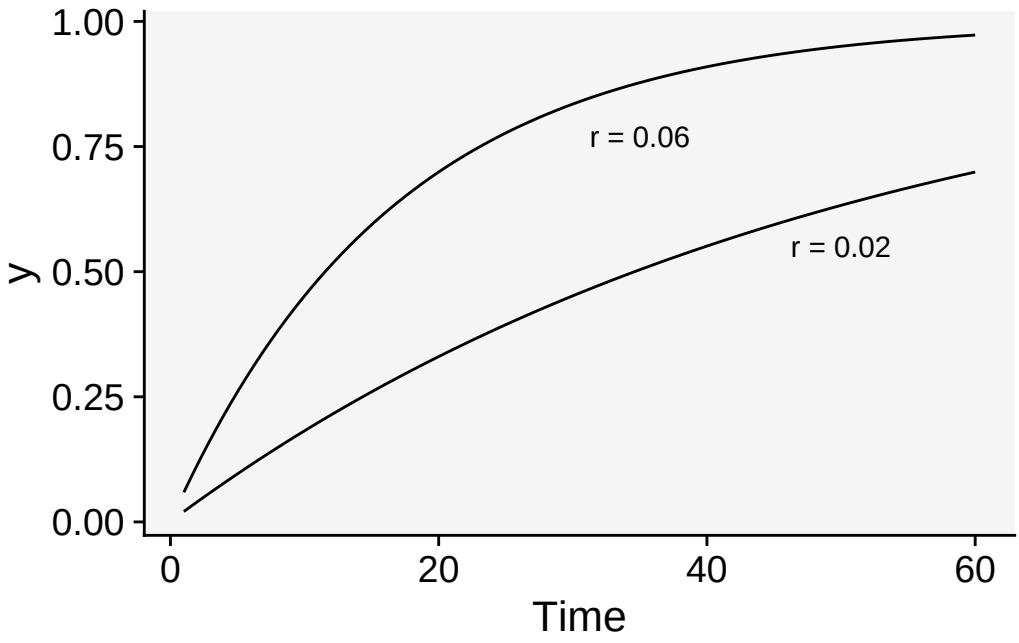


Figure 11.3: Monomolecular curves with two rates of infection (0.06 and 0.02) and the same initial inoculum (0.001)

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))) +
  theme_r4pde()+
  labs(x = "Time") +
  annotate(geom = "text", x = 35, y = 0.77, label = "y0 = 0.1") +
  annotate(geom = "text", x = 45, y = 0.65, label = "y0 = 0.001")
```

11.1.3 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.

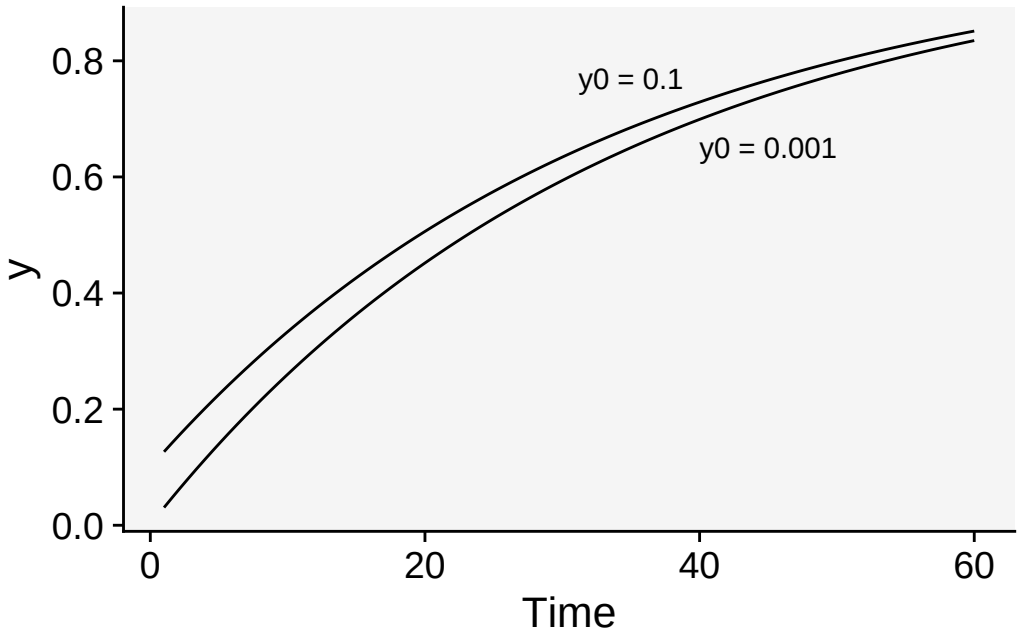


Figure 11.4: Monomolecular curves with one rate (0.06) and the initial inoculum increased 100 times

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

The integral is given by

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

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

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

```
dat |>
  ggplot(aes(t, y)) +
  stat_function(
    linetype = 2,
    fun = function(t) 1 / (1 + ((1 - y0) / y0) * exp(-r * 2 * t))
```

```
  ) +
  stat_function(fun = function(t) 1 / (1 + ((1 - y0) / y0) * exp(-r * 4 * t))) +
  labs(x = "Time") +
theme_r4pde()+
  annotate(geom = "text", x = 41, y = 0.77, label = "r = 0.18") +
  annotate(geom = "text", x = 50, y = 0.10, label = "r = 0.024")
```

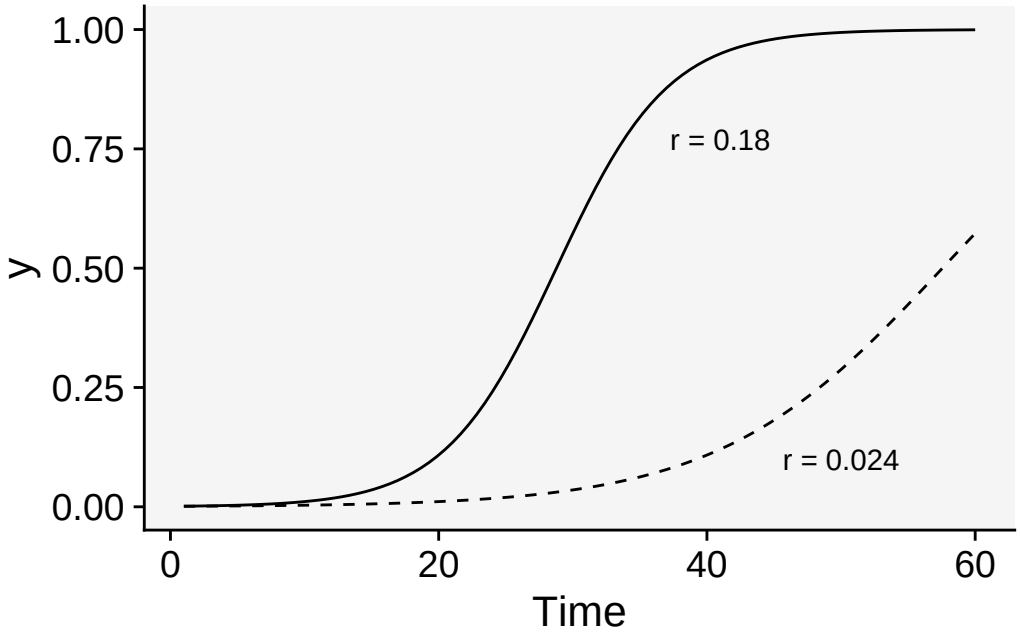


Figure 11.5: Logistic curves with two rates of infection (0.18 and 0.024) and the same initial inoculum (0.001)

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

```
dat |>
  ggplot(aes(t, y)) +
  stat_function(
    linetype = 2,
    fun = function(t) 1 / (1 + ((1 - (y0 / 10)) / (y0 / 10)) * exp(-r * 3 * t))
  ) +
  stat_function(fun = function(t) 1 / (1 + ((1 - y0) / y0) * exp(-r * 3 * t))) +
  labs(x = "Time") +
 theme_r4pde()+
  annotate(geom = "text", x = 35, y = 0.77, label = "y0 = 0.001") +
```

```
  annotate(geom = "text", x = 50, y = 0.10, label = "y0 = 0.0001")
```

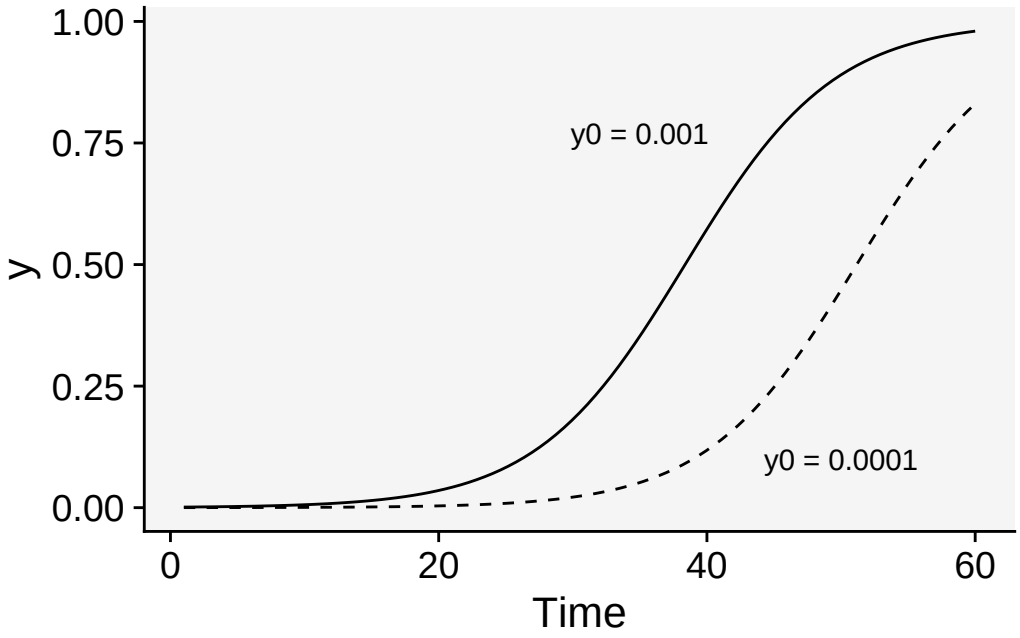


Figure 11.6: Logistic curves with a single rate of infection (0.24) and two initial inoculum (0.001 and 0.0001)

11.1.4 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(x = "Time") +
 theme_r4pde()+
  annotate(geom = "text", x = 41, y = 0.77, label = "r = 0.12") +
  annotate(geom = "text", x = 50, y = 0.10, label = "r = 0.03")
```

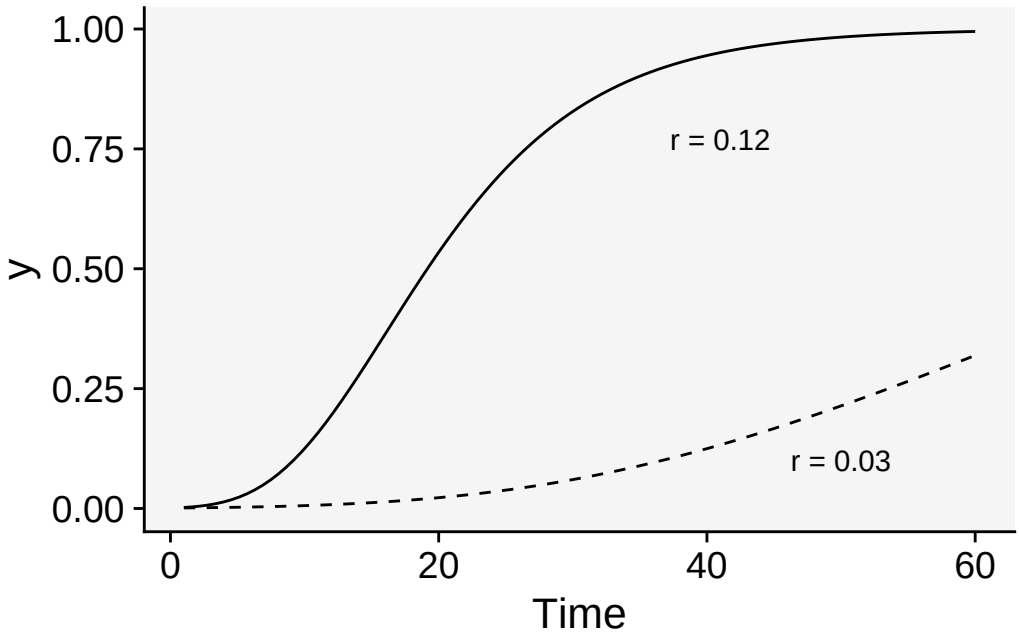


Figure 11.7: Gompertz curves with two rates of infection (0.12 and 0.03) and the same initial inoculum (0.001)

And those when inoculum was reduced one 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(x = "Time") +
theme_r4pde()+
  annotate(geom = "text", x = 15, y = 0.77, label = "y0 = 0.001") +
  annotate(geom = "text", x = 25, y = 0.10, label = "y0 = 0.00001")
```

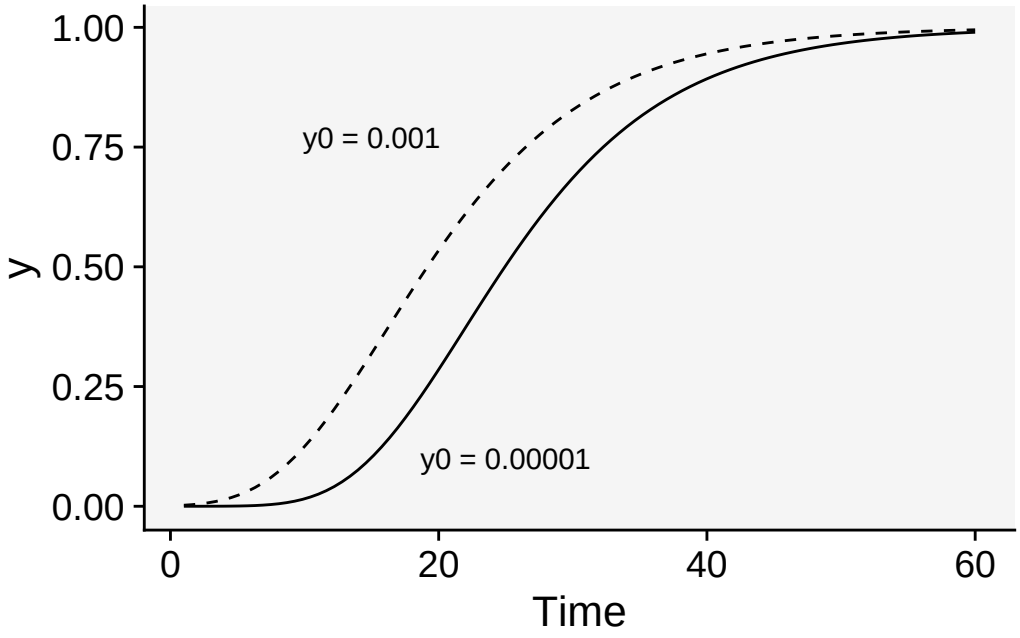


Figure 11.8: Gompertz curves with a single rate of infection (0.12) and two levels of initial inoculum (0.001 and 0.00001)

11.2 Interactive application

A shiny app was developed to demonstrate these four models interactively. Click on the image below to get access to the app.

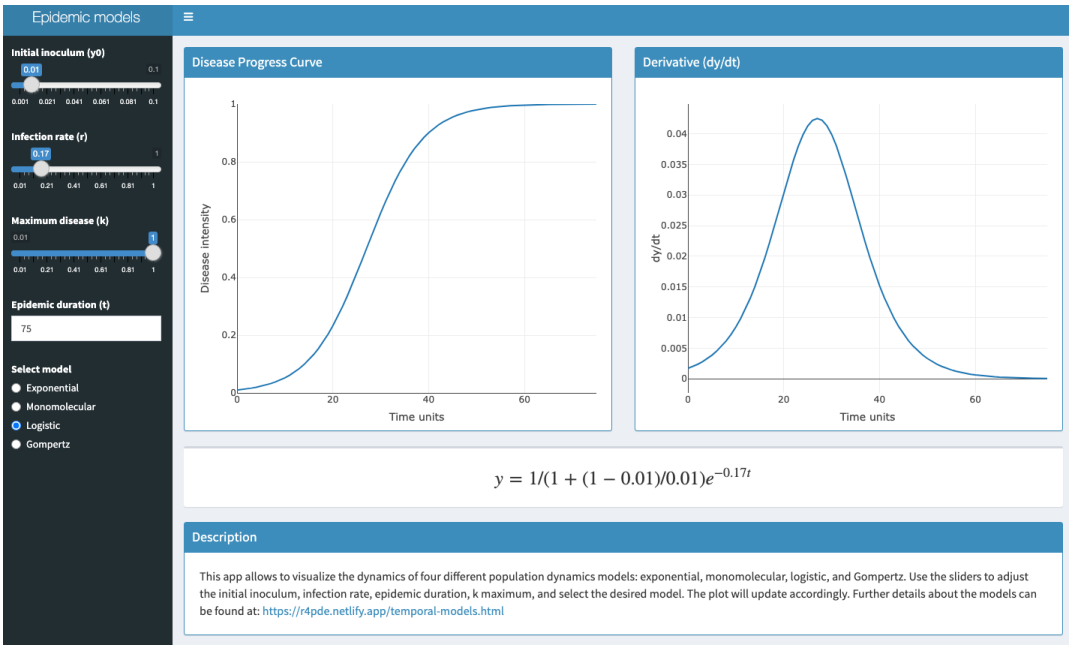


Figure 11.9: Screenshot of the application to visualize the population dynamics models by varying the model's parameters

12 Model fitting

In model fitting for temporal analysis, the objective is to determine which previously-reviewed epidemiological (population dynamics) models best fit the data from actual epidemics. Doing so allows us to obtain two key parameters: the initial inoculum and the apparent infection rate.

There are essentially two methods for achieving this: linear regression and non-linear regression modeling. We'll begin with linear regression, which is computationally simpler. I'll illustrate the procedure using both built-in R functions and custom functions from the epifitter package (Alves and Del Ponte 2021). Epifitter offers a set of user-friendly functions that can fit and rank the best models for a given epidemic.

To exemplify, we'll continue examining a previously shown curve that represents the incidence of the tobacco etch virus, a disease affecting peppers, over time. This dataset is featured in Chapter 3 of the book, "Study of Plant Disease Epidemics" (Madden et al. 2007c). While the book presents SAS code for certain analyses, we offer an alternative code that accomplishes similar analyses, even if it doesn't replicate the book's results exactly.

12.1 Linear regression: single epidemics

```
library(tidyverse)
library(r4pde)
theme_set(theme_r4pde())
dpc <-
  tribble(
   ~t,  ~y,
   0,  0.1,
   7,  1,
  14,  9,
  21,  25,
  28,  80,
  35, 98,
  42, 99,
  49, 99.9
```

```
  )
```

```
dpc |>
  ggplot(aes(t, y))+
  geom_point(size =3)+
  geom_line()+
  theme_r4pde()+
  labs(x = "Time", y = "Disease intensity (%)")
```

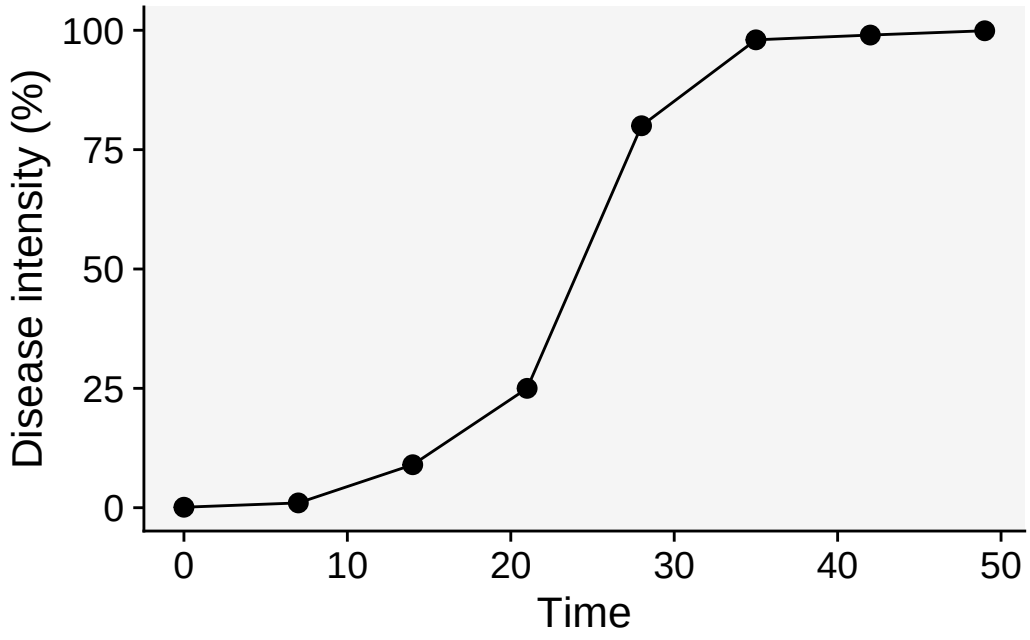


Figure 12.1: Disease progress curves for one tobacco etch epidemics in pepper. Reproduced from Madden et al. (2007c) page 94

To start, we'll need to transform the disease intensity (in proportion scale) data according to each of the models we aim to fit. In this instance, we'll look at the four models discussed in the previous chapter: exponential, monomolecular, logistic, and Gompertz. We can use the `mutate()` function of *dplyr* package. The transformed y will be referred to as y* (or y2 in the code) followed by the letter E, M, L or G, for each model (exponential, monomolecular, etc) respectively.

```
dpc1 <- dpc |>
  mutate(y = y/100) |> # transform to proportion
  mutate(exponential = log(y),
```

```
         monomolecular = log(1 / (1 - y)),
         logistic = log(y / (1 - y)),
         gompertz = -log(-log(y)))
knitr::kable(round(dpc1, 4))
```

t	y	exponential	monomolecular	logistic	gompertz
0	0.001	-6.9078	0.0010	-6.9068	-1.9326
7	0.010	-4.6052	0.0101	-4.5951	-1.5272
14	0.090	-2.4079	0.0943	-2.3136	-0.8788
21	0.250	-1.3863	0.2877	-1.0986	-0.3266
28	0.800	-0.2231	1.6094	1.3863	1.4999
35	0.980	-0.0202	3.9120	3.8918	3.9019
42	0.990	-0.0101	4.6052	4.5951	4.6001
49	0.999	-0.0010	6.9078	6.9068	6.9073

Now we can plot the curves using the transformed values regressed against time. The curve that appears most linear, closely coinciding with the regression fit line, is a strong candidate for the best-fitting model. To accomplish this, we'll first reshape the dataframe into long format, and then generate plots for each of the four models.

```
dpc2 <- dpc1 |>
  pivot_longer(3:6, names_to = "model", values_to = "y2")


dpc2 |>
  ggplot(aes(t, y2))+
  geom_point()+
  geom_smooth(method = "lm", color = "black", se = F)+
  facet_wrap(~ model)+
  theme_r4pde()+
  labs(x = "Time", y = "Transformed value (y*)",
       color = "Model")+
  theme(legend.position = "none")
```

For this particular curve, it's readily apparent that the logistic model offers the best fit to the data, as evidenced by the data points being closely aligned with the regression line, compared to the other models. However, to make a more nuanced decision between the logistic and Gompertz models—which are both typically used for sigmoid curves—we can rely on additional statistical measures.

Specifically, we can fit a regression model for each and examine key metrics such as the R-

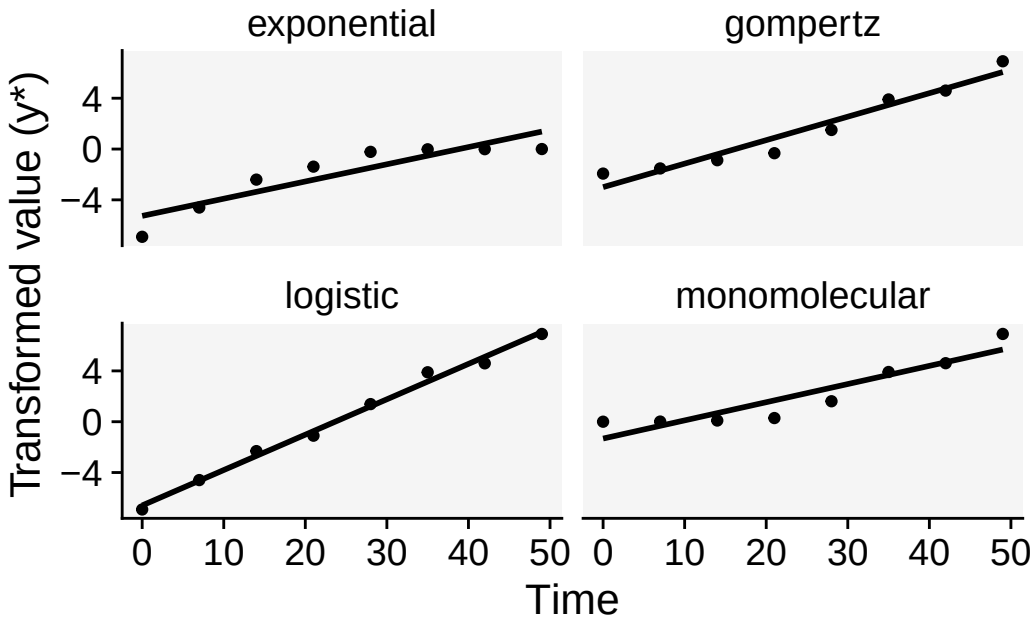


Figure 12.2: Curves of the transformed data for each epidemiological against time. The goal is to check which of the models provides the best fit based on the straight line

squared value and the residual standard error. To further validate the model's accuracy, we can use Lin's Concordance Correlation Coefficient to assess how closely the model's predictions match the actual (transformed) data points.

For this exercise, let's focus on the logistic and Gompertz models. We'll start by fitting the logistic model and then move on to analyzing the summary of the regression model.

```
logistic <- dpc2 |>
  filter(model == "logistic")

m_logistic <- lm(y2 ~ t, data = logistic)


# R-squared
summary(m_logistic)$r.squared
```

```
[1] 0.9923659
```

```r
# RSE
summary(m_logistic)$sigma
```

```
[1] 0.4523616
```

```r
# calculate the Lin's CCC
library(epiR)
ccc_logistic <- epi.ccc(logistic$y2, predict(m_logistic))
ccc_logistic$rho.c[1]
```

```
       est
1 0.9961683
```

We repeat the procedure for the Gompertz model.

```r
gompertz <- dpc2 |>
  filter(model == "gompertz")

m_gompertz <- lm(y2 ~ t, data = gompertz)

# R-squared
summary(m_gompertz)$r.squared
```

```
[1] 0.9431066
```

```r
# RSE
summary(m_gompertz)$sigma
```

```
[1] 0.8407922
```

```r
# calculate the Lin's CCC
library(epiR)
ccc_gompertz <- epi.ccc(gompertz$y2, predict(m_gompertz))
ccc_gompertz$rho.c[1]
```

```
       est
1 0.9707204
```

Next, let's extract the two parameters of interest from each fitted model and incorporate them into the integral form of the respective models. To do this, we'll need to back-transform the intercept, which represents the initial inoculum. This can be accomplished using specific equations, which we'll outline next.

Model	Transformation	Back-transformation
Exponential	log(y)	exp(y*E)
Monomolecular	log(1 / (1 - y))	1 - exp(-y*M)
Logistic	log(y / (1 - y))	1 / (1 + exp(-y*L))
Gompertz	-log(-log(y))	exp(-exp(-y*G))

```
rL <- m_logistic$coefficients[2]
rL
```

```
        t
0.2784814
```

```
y02 <- m_logistic$coefficients[1]
y0L = 1 / (1 + exp(-y02))
y0L
```

```
(Intercept)
0.001372758
```

```
rG <-m_gompertz$coefficients[2]
rG
```

```
        t
0.1848378
```

```
y03 <- m_gompertz$coefficients[1]
y0G <- exp(-exp(-y03))
y0G
```

```
 (Intercept)
1.968829e-09
```

Now the plot:

```
logistic |>
  ggplot(aes(t, y)) +
  geom_point(size = 2)+
  stat_function(
    linetype = 2,
    fun = function(t) 1 / (1 + ((1 - y0L) / y0L) * exp(-rL * t)))+
stat_function(
    linetype = 1,
    fun = function(t) exp(log(y0G) * exp(-rG * t))
  )+
  theme_r4pde()+
  labs(x = "Time", y = "Disease intensity")
```

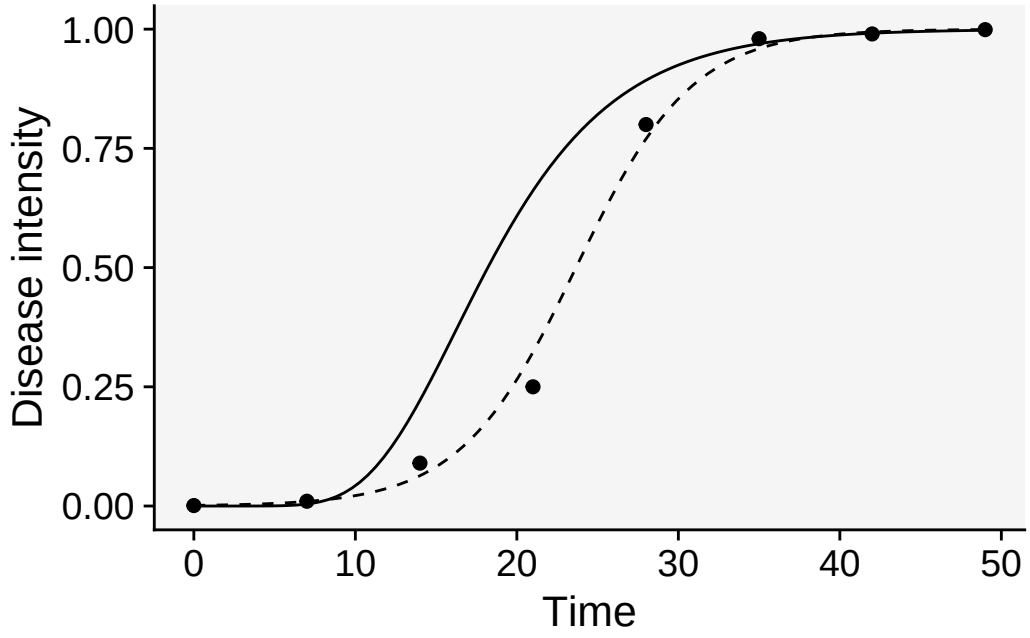


Figure 12.3: Disease progress curve and the fit of the logistic (dashed line) and the Gompertz (solid line) based on parameters estimated using linear regression

In this case, it's clear that the logistic model (the solid line above) emerges as the best fit based on our statistical evaluation. The approach for model selection outlined here is straightforward and manageable when dealing with a single epidemic and comparing only two models. However, real-world scenarios often require analyzing multiple curves and fitting various models to each, making manual comparison impractical for selecting a single best-fitting model. To streamline this task, it's advisable to automate the process using custom functions designed to simplify the coding work involved.

That's where the epifitter package comes into play! This package offers a range of custom functions designed to automate the model fitting and selection process, making it much more efficient to analyze multiple curves across different epidemics. By using epifitter, one can expedite the statistical evaluation needed to identify the best-fitting models.

12.2 Non linear regression

Alternatively, one can fit a nonlinear model to the data for each combination of curve and model using the nlsLM function in R of the *minpack.lm* package.

```
library(minpack.lm)
fit_logistic <- nlsLM(y/100 ~ 1 / (1+(1/y0-1)*exp(-r*t)),
            start = list(y0 = 0.01, r = 0.3),
            data = dpc)

summary(fit_logistic)
```

```
Formula: y/100 ~ 1/(1 + (1/y0 - 1) * exp(-r * t))

Parameters:
    Estimate Std. Error t value Pr(>|t|)
y0 0.0003435  0.0002065   1.663    0.147
r  0.3321352  0.0249015  13.338  1.1e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.02463 on 6 degrees of freedom

Number of iterations to convergence: 15
Achieved convergence tolerance: 1.49e-08
```

```
fit_gompertz <- nlsLM(y/100 ~ exp(log(y0/1)*exp(-r*t)),
                    start = list(y0 = 0.01, r = 0.1),
                    data = dpc)
summary(fit_gompertz)
```

```
Formula: y/100 ~ exp(log(y0/1) * exp(-r * t))
```

```
Parameters:
    Estimate Std. Error t value Pr(>|t|)
y0 2.038e-15  4.950e-14   0.041  0.96850
r  1.643e-01  3.033e-02   5.416  0.00164 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.06691 on 6 degrees of freedom

Number of iterations till stop: 50
Achieved convergence tolerance: 1.49e-08
Reason stopped: Number of iterations has reached `maxiter' == 50.
```

We can see that the model coefficients are not the same as those estimated using linear regression. Among other reasons, `nls()` often uses iterative techniques to estimate parameters, such as the Levenberg-Marquardt algorithm, which may provide different estimates than algebraic methods used in linear regression. While both methods aim to fit a model to data, they do so in ways that have distinct assumptions, strengths, and weaknesses, and this can result in different estimated parameters.

Both approaches—nonlinear least squares and linear regression on transformed data—have their own merits and limitations. The choice between the two often depends on various factors like the nature of the data, the underlying assumptions, and the specific requirements of the analysis. For an epidemiologist, the choice might come down to preference, familiarity with the techniques, or specific aims of the analysis.

In summary, both methods are valid tools in the toolkit of an epidemiologist or any researcher working on curve fitting and model selection. Understanding the nuances of each can help in making an informed choice tailored to the needs of a particular study.

12.3 epifitter - multiple epidemics

We will now examine three disease progress curves (DPCs) representing the incidence of the tobacco etch virus, a disease affecting peppers. Incidence evaluations were conducted at 7-day intervals up to 49 days. The relevant data can be found in Chapter 4, page 93, of the book "Study of Plant Disease Epidemics" (Madden et al. 2007c). To get started, let's input the data manually and create a data frame. The first column will represent the assessment time, while the remaining columns will correspond to the treatments, referred to as 'groups' in the book, ranging from 1 to 3.

12.4 Entering data

```
library(tidyverse) # essential packages
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
  )
```

12.5 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. (2007c) 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 DPC is in a different column) 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))+
  scale_color_grey()+
  theme_grey()+
  geom_line(linewidth = 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_r4pde()+
theme(legend.position = "none")
```

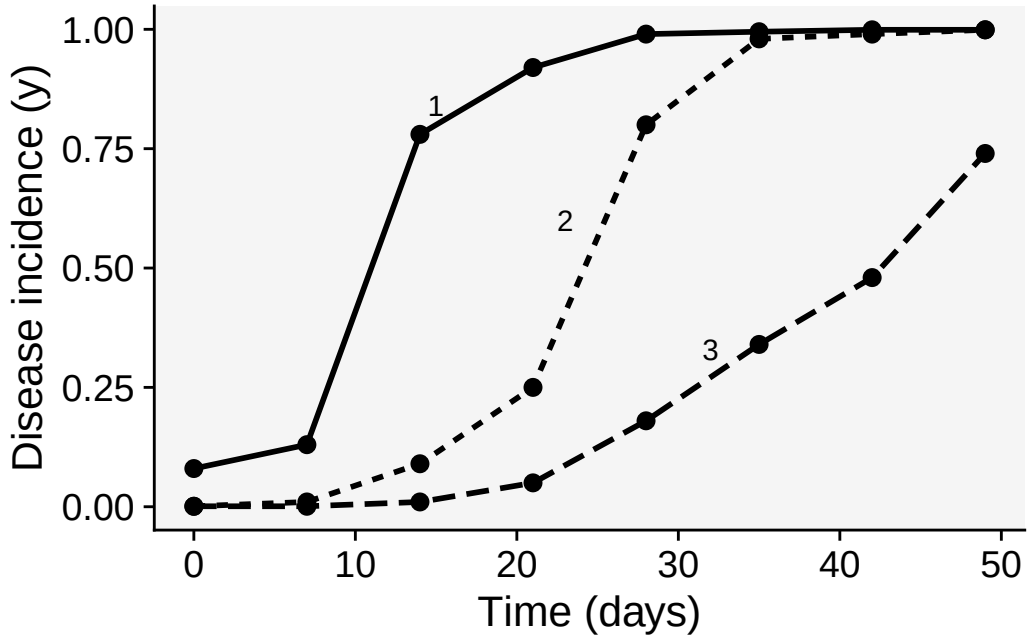


Figure 12.4: Disease progress curves for three tobacco etch epidemics in pepper. Reproduced from Madden et al. (2007c) page 94

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.

12.5.1 epifitter: linear regression

Among the several options offered by *epifitter* we start with the simplest one, which is to 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` )
knitr::kable(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` )
knitr::kable(epi2$Stats)
```

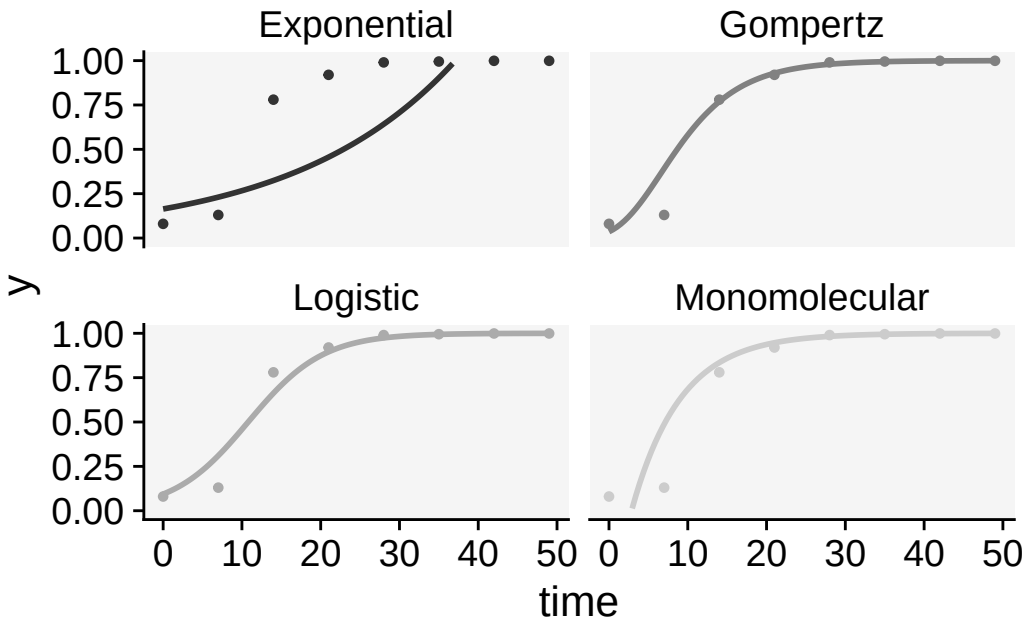
	CCC	r_squared	RSE
Logistic	0.9962	0.9924	0.4524
Gompertz	0.9707	0.9431	0.8408
Monomolecular	0.9248	0.8601	1.0684
Exponential	0.8971	0.8134	1.2016

```
epi3 <- fit_lin(time = pepper$t,
  y = pepper$`3` )
knitr::kable(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 a slight 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. For such, we can use the `plot_fit()` function from *epifitter* to explore visually the fit of the four models to each curve.

```
plot_fit(epi1)+
  ylim(0,1)+
  scale_color_grey()+
  theme_r4pde()+
  theme(legend.position = "none")
```

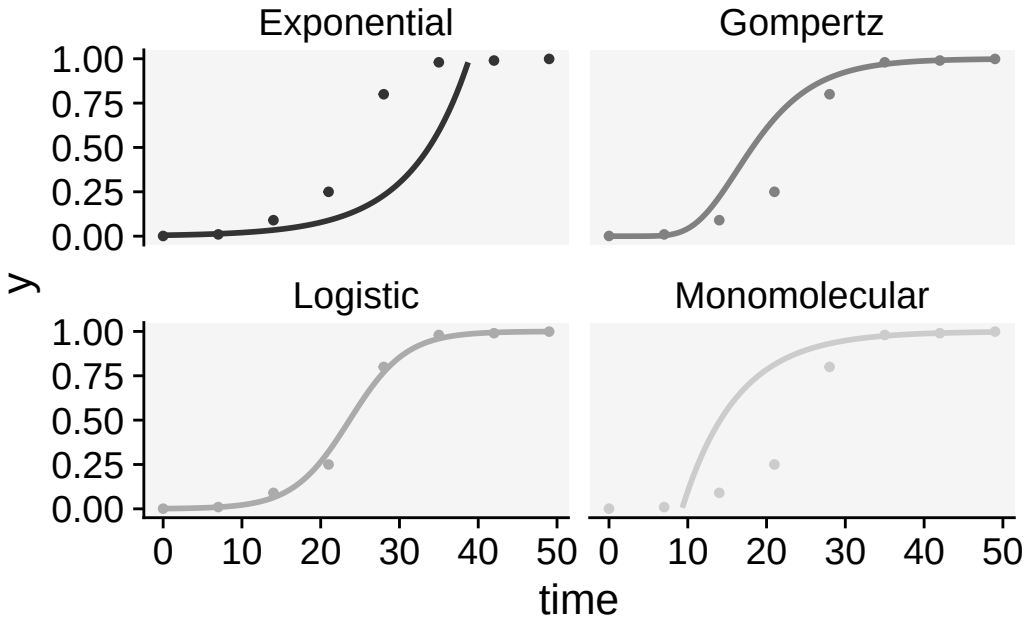



```
knitr::kable(epi1$stats_all)
```

best_model	model	r	r_se	r_ci_lwr	r_ci_upr	y0	y0_se	r_square	RSE	CCC	y0	y0_ci_lwr	y0_ci_upr
1	Gompertz	0.1815703	0.0130299	0.1549688	0.2134544	-1.2050364	0.3815579	0.9700027	0.2739110	0.9847857	0.9735547	0.9700205	0.9769349
2	Monomolecular	0.1616401	0.0119733	0.1382340	0.1909404	-0.4625249	0.3506326	0.9681255	0.2543197	0.9838044	-0.5880787	-0.7452636	-0.3266178
3	Logistic	0.2104047	0.0181544	0.1659822	0.2548270	-2.2715851	0.5316185	0.9572418	0.2357798	0.9781534	0.9350382	0.9273207	0.9427287

best_model	model	r	r_se	r_ci_lwr	r_ci_upr	v0	v0_se	r_squared	RSE	CCC	y0	y0_ci_lwr	y0_ci_upr
4	Exponential	0.1187634	0.1478021	0.0125974	0.4849293	-1.8090602	0.4328101	0.1344653	0.1705087	0.0783938	0.1638088	0.0568064	0.4723623

```
plot_fit(epi2)+
  ylim(0,1)+
  scale_color_grey()+
  scale_color_grey()+
  theme_r4pde()+
  theme(legend.position = "none")
```

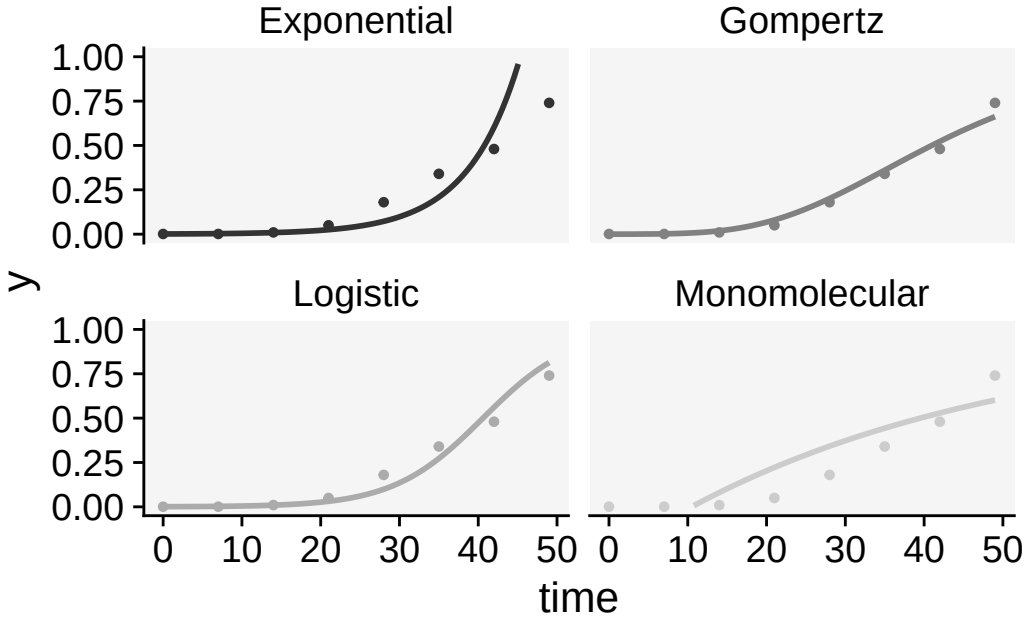



```
knitr::kable(epi2$stats_all)
```

best_model	model	r	r_se	r_ci_lwr	r_ci_upr	v0	v0_se	r_squared	RSE	CCC	y0	y0_ci_lwr	y0_ci_upr
1	Logistic	0.2784814	0.1409971	0.0254081	0.5028809	-6.589560	0.2919989	0.0923654	0.1523619	0.0961683	0.0137288	0.0006724	0.4028007
2	Gompertz	0.1848378	0.1853933	0.0394872	0.4301886	-2.998021	0.5427299	0.0431068	0.1640792	0.0270720	0.0000000	0.0000000	0.0049309
3	Monomolecular	0.1430234	0.2355003	0.0085397	0.7006489	-1.325645	0.6896258	0.0600832	0.1526836	0.0247793	-2.7646136	-6.3503499	0.3035837

best_model	model	r	r_se	r_ci_lwr	r_ci_upr	v0	v0_se	r_squared	RSE	CCC	y0	y0_ci_lwr	y0_ci_upr
4	Exponential	0.1354579	0.0264869	0.0970646	0.2002689	-5.263915	0.7756178	0.7134012	0.1201580	0.8971003	0.0051750	0.0007757	0.0345258

```
plot_fit(epi3)+
  ylim(0,1)+
  scale_color_grey()+
  scale_color_grey()+
  theme_r4pde()+
  theme(legend.position = "none")
```




```
knitr::kable(epi3$stats_all)
```

best_model	model	r	r_se	r_ci_lwr	r_ci_upr	v0	v0_se	r_squared	RSE	CCC	y0	y0_ci_lwr	y0_ci_upr
1	Logistic	0.1752104	0.0613325	0.0774260	0.3778215	-7.1136060	0.3902189	0.8766459	0.0960452	0.9382943	0.0008138	0.0003132	0.0021104
2	Gompertz	0.0647104	0.0504987	0.0145251	0.2769182	-2.2849079	0.1460479	0.6558922	0.2625598	0.8249350	0.0005410	0.0000018	0.0010358
3	Exponential	0.1513089	0.0169861	0.0975569	0.2288228	-6.8629493	0.4974039	0.9297097	0.0770573	0.9635747	0.0010458	0.0003097	0.0035322

best_model	model	r	r_se	r_ci_lwr	r_ci_upr	v0	v0_se	r_squared	RSE	CCC	y0	y0_ci_lwr	y0_ci_upr
4	Monomolecular	0.0238057	0.0055853	0.0102903	0.0375624	-0.2506567	0.1635537	0.7531302	0.2533768	0.8591837	-0.2848689	-0.3171854	0.1389001

12.5.2 epifitter: non linear regression

```
epi11 <- fit_nlin(time = pepper$t,
                  y = pepper$`1` )
knitr::kable(epi11$Stats)
```

	CCC	r_squared	RSE
Gompertz	0.9963	0.9956	0.0381
Logistic	0.9958	0.9939	0.0403
Monomolecular	0.9337	0.8883	0.1478
Exponential	0.7161	0.5903	0.2770

```
epi22 <- fit_nlin(time = pepper$t,
                  y = pepper$`2` )
knitr::kable(epi22$Stats)
```

	CCC	r_squared	RSE
Logistic	0.9988	0.9981	0.0246
Gompertz	0.9904	0.9857	0.0683
Monomolecular	0.8697	0.8020	0.2329
Exponential	0.8587	0.7862	0.2413

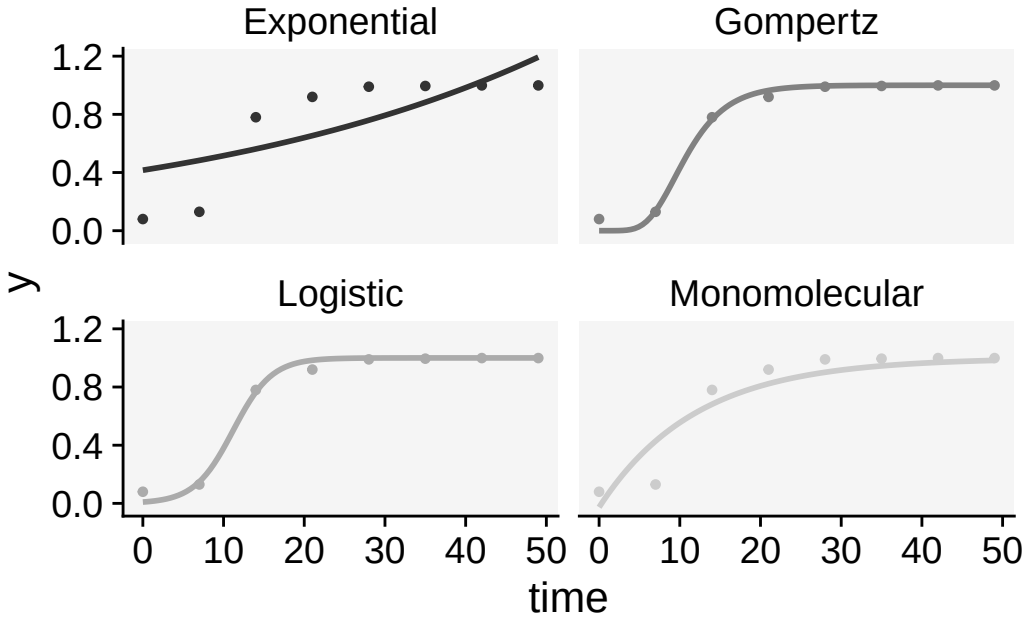
```
epi33 <- fit_nlin(time = pepper$t,
                  y = pepper$`3` )
knitr::kable(epi33$Stats)
```

	CCC	r_squared	RSE
Logistic	0.9957	0.9922	0.0270
Gompertz	0.9946	0.9894	0.0306
Exponential	0.9880	0.9813	0.0445
Monomolecular	0.8607	0.7699	0.1426

And now we can produce the plot of the fitted curves together with the original incidence dat. The `stats_all` dataframe shows everything we need regarding the statistics and the values of the parameteres.

```
plot_fit(epi11)+
  scale_color_grey()+
  scale_color_grey()+
  theme_r4pde()+
  theme(legend.position = "none")
```

```
Scale for colour is already present.
Adding another scale for colour, which will replace the existing scale.
```



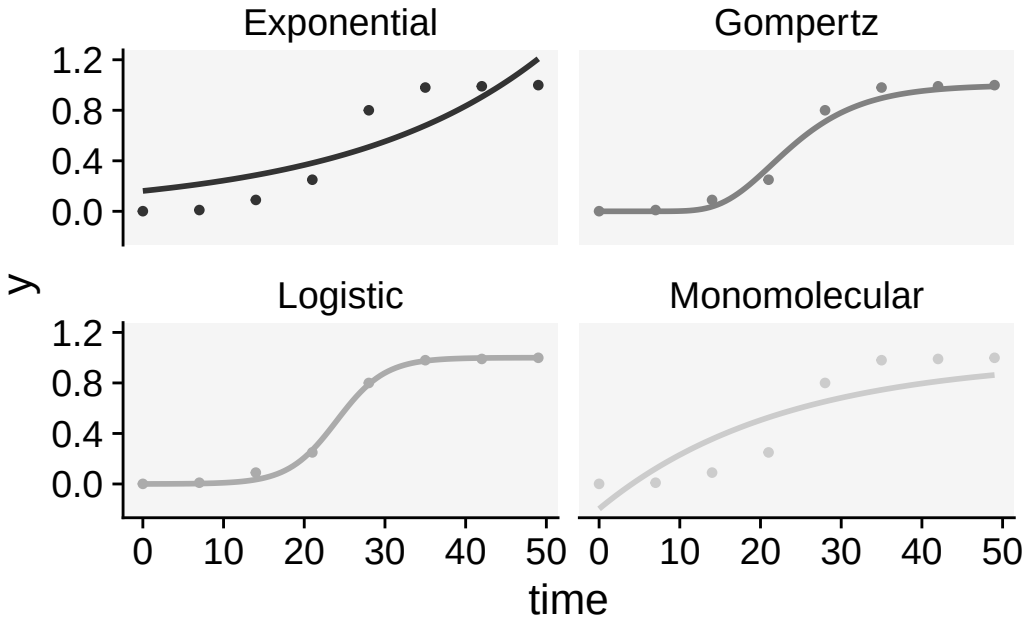

```
knitr::kable(epi11$stats_all)
```

model	y0	y0_se	r	r_se	df	CCC	r_squared	RSE	y0_ci_lwr	y0_ci_upr	r_ci_lwr	r_ci_upr	best_model
Gompertz	0.0000004	0.0000012	0.1786502	0.0031316	59	0.9962708	0.9956357	0.0380871	-0.0000039	0.0000042	0.1709875	0.1863130	1
Logistic	0.0092495	0.0060334	0.1818135	0.0053996	61	0.9957911	0.9938810	0.0403179	-0.0055126	0.0240117	0.1708601	0.1927669	2

model	y0	y0_se	r	r_se	df	CCC	r_squared	RSE	y0_ci_lwr	y0_ci_upr	r_ci_lwr	r_ci_upr	best_model
Monomolecular	0.0264718	0.1405690	0.0836206	0.0212467	3	0.9337218	0.8883221	0.0477596	-0.3704319	0.3748830	0.0316303	0.1356109	3
Exponential	0.1159601	0.0613767	0.0621506	0.0088658	4	0.7160765	0.5903402	0.0770420	-0.0790782	0.2528430	0.0432000	0.0001874	4

```
plot_fit(epi22)+
  scale_color_grey()+
  scale_color_grey()+
  theme_r4pde()+
  theme(legend.position = "none")
```

```
Scale for colour is already present.
Adding another scale for colour, which will replace the existing scale.
```

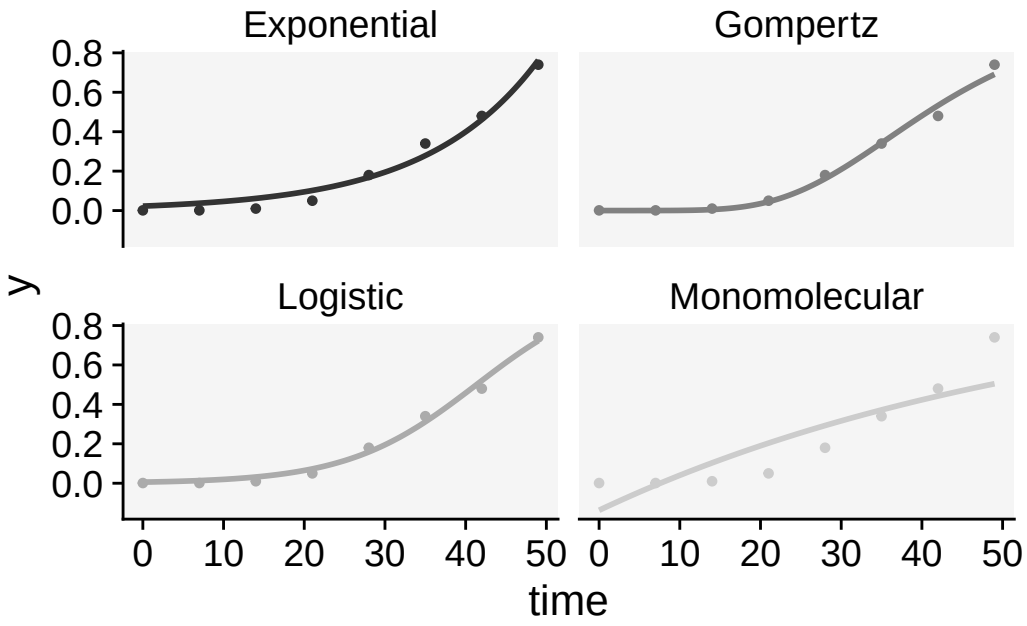



```
knitr::kable(epi22$stats_all)
```

model	y0	y0_se	r	r_se	df	CCC	r_squared	RSE	y0_ci_lwr	y0_ci_upr	r_ci_lwr	r_ci_upr	past_model
Logistic	0.0003435	0.0002063	0.3321341	0.0249061	14	0.9988259	0.9980699	0.0246349	-0.0001618	0.0008488	0.2871202	0.3793065	1
Gompertz	2.000000	0.0000000	0.1061874	0.0301532	15	0.9904459	0.9856899	0.0682500	0.0000000	0.0000000	0.0880938	0.1235654	2
Monomolecular	0.2005248	0.1971531	0.0601316	0.0442066	14	0.8696738	0.7560352	0.2328814	-0.0935130	0.6878195	-0.0119840	0.1764280	3
Exponential	0.1612234	0.0848398	0.0410930	0.0125521	15	0.8587177	0.7686204	0.2412526	-0.0688180	0.3913779	0.0103707	0.0718076	4

```
plot_fit(epi33)+
  scale_color_grey()+
  scale_color_grey()+
  theme_r4pde()+
  theme(legend.position = "none")
```

```
Scale for colour is already present.
Adding another scale for colour, which will replace the existing scale.
```




```
knitr::kable(epi33$stats_all)
```

model	y0	y0_se	r	r_se	df	CCC	r_squared	RSE	y0_ci_lwr	y0_ci_upr	r_ci_lwr	r_ci_upr	best_model
Logistic	0.0056634	0.0019662	0.1249903	0.0108671	11	0.9957129	0.9922359	0.5927031	0.0010852	0.0102417	0.1037758	0.1462105	1
Gompertz	0.0000002	0.0000008	0.0875983	0.0061025	11	0.9945749	0.9894187	0.7305883	-0.0000017	0.0000024	0.0610515	0.1909159	2
Exponential	0.0225267	0.0072188	0.1718824	0.0070479	11	0.9880169	0.9812850	0.9445057	0.0070486	0.0380189	0.1546367	0.1891281	3
Monomolecular	0.1060292	0.0169600	0.0042530	0.8606957	0.7698769	1.9426044	-0.3965926	0.1371485	0.1222956	0.0065530	0.0273669	4	

For multiple epidemics, we can 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. To use the nonlinear regression approach we set `nlin` argument to `TRUE`.

```
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
```

The code below calculates the frequency that each model was the best. This would facilitate in the case of many epidemics to analyse.

```
freq_best <- epi_all$Parameters %>%
    filter(best_model == 1) %>%
    group_by(treat, model) %>%
    summarise(first = n()) %>%
  ungroup() |>
  count(model)
freq_best
```

```
# A tibble: 2 x 2
  model        n
  <chr>    <int>
1 Gompertz     1
2 Logistic     2
```

We can see that the Logistic model was the best model in two out of three epidemics.

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) +
  scale_color_grey()+
  theme_r4pde()+
  theme(legend.position = "bottom")+
```

```
coord_cartesian(ylim = c(0, 1)) + # set the max to 0.6
 labs(
   shape = "Epidemic",
   y = "Disease incidence",
   x = "Time (days after emergence)"
 )
```

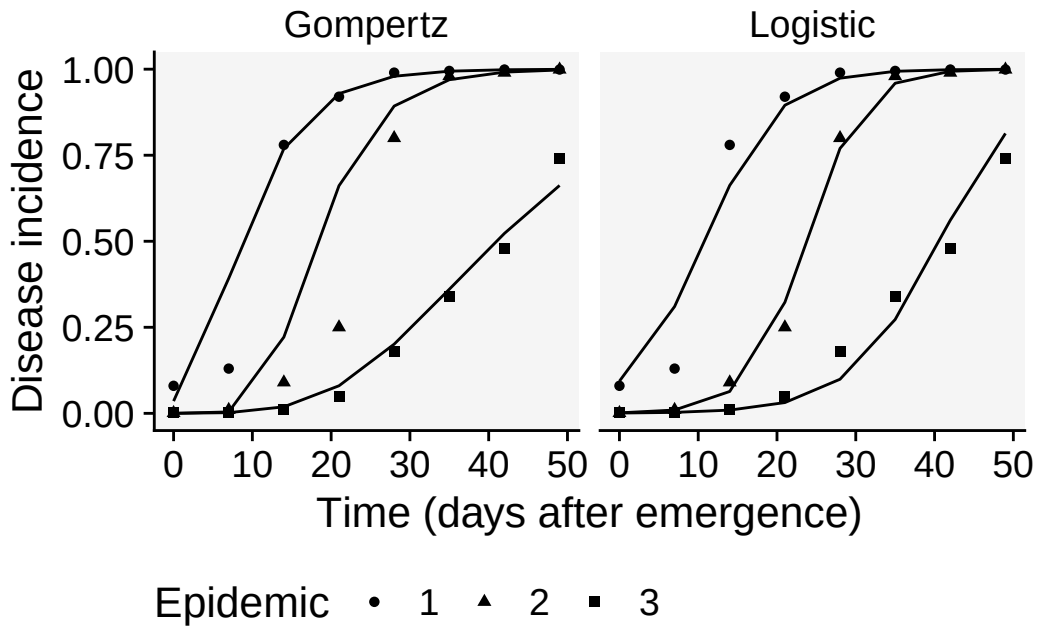


Figure 12.5: Observed (dots) and fitted (line) values for three tobacco etch epidemics in pepper

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)) +
  scale_color_grey()+
  theme_r4pde()+
  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)",
    shape = "Epidemic"
  )
```

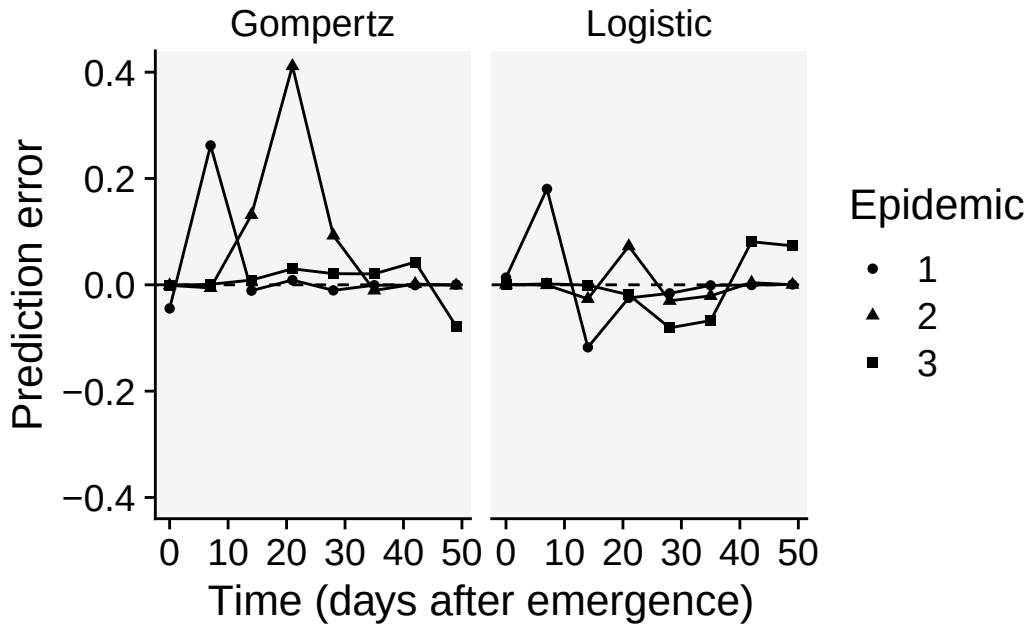


Figure 12.6: Prediction error (dotted lines) by two models fitted to the progress curves of three tobacco etch epidemics in pepper

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.

We can then now extract the parameters of interest of the chosen model. These data are stored in the `Parameters` data frame of the `epi_all` list. Let's filter the Logistic model and apply a selection of the parameters of interest.

```
  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.

```
p1 <- epi_all$Parameters |>
  filter(model == "Logistic") |>
  ggplot(aes(treat, r)) +
  scale_color_grey()+
  theme_r4pde()+
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = r_ci_lwr, ymax = r_ci_upr),
    width = 0,
    size = 1
  ) +
  labs(
    x = "Epidemic",
    y = "r"
  )
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```

```
p2 <- epi_all$Parameters |>
  filter(model == "Logistic") |>
  ggplot(aes(treat, 1 - exp(-y0))) +
  scale_color_grey()+
  theme_r4pde()+
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = y0_ci_lwr, ymax = y0_ci_upr),
    width = 0,
    size = 1
  ) +
  labs(
    x = "Epidemic",
    y = "y0"
  )

library(patchwork)
p1 | p2
```

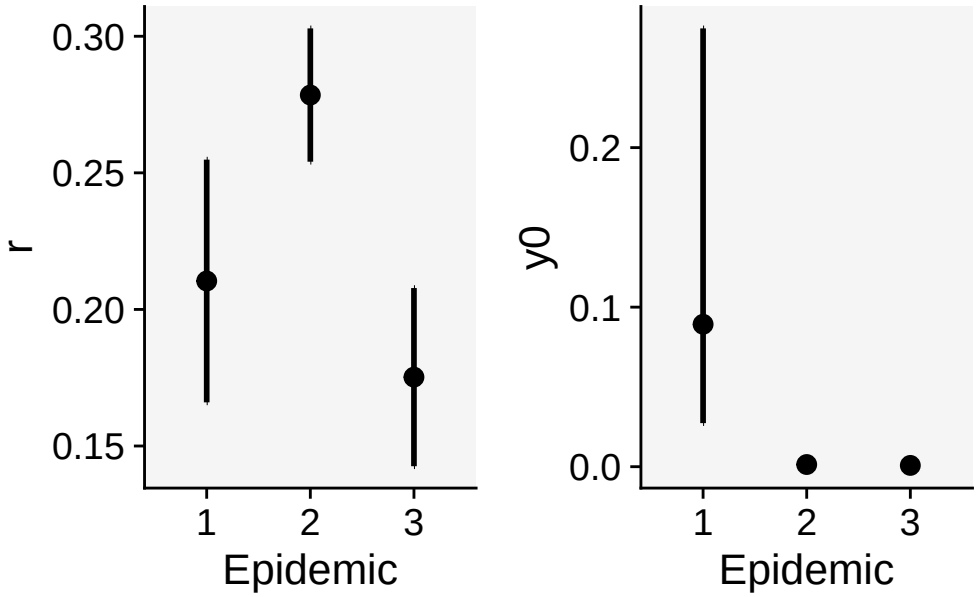


Figure 12.7: Estimated infection rates (left) and initial inoculum (right) by a logistic model fitted to the progress curves of three epidemics of tobacco etch on pepper

We can compare the rate parameter (slopes) from two separate linear regression models using a t-test. This is sometimes referred to as a "test of parallelism" in the context of comparing slopes. The t-statistic for comparing two slopes with their respective standard errors can be calculated as:

$t = \frac{\beta_1 - \beta_2}{\sqrt{SE_{\beta_1}^2 + SE_{\beta_2}^2}}$

This t-statistic follows a t-distribution with (df = n_1 + n_2 - 4) degrees of freedom, where (n_1) and (n_2) are the sample sizes of the two groups. In our case, (n_1 = n_2 = 8), so (df = 8 + 8 - 4 = 12).

Here's how to perform the t-test for comparing curve 1 and 2.

```
# Given slopes and standard errors from curve 1 and 2
beta1 <- 0.2104
beta2 <- 0.2784
SE_beta1 <- 0.01815
SE_beta2 <- 0.00997

# Sample sizes for both treatments (n1 and n2)
```

```r
n1 <- 8
n2 <- 8

# Calculate the t-statistic
t_statistic <- abs(beta1 - beta2) / sqrt(SE_beta1^2 + SE_beta2^2)

# Degrees of freedom
df <- n1 + n2 - 4

# Calculate the p-value
p_value <- 2 * (1 - pt(abs(t_statistic), df))

# Print the results
print(paste("t-statistic:", round(t_statistic, 4)))
```

```
[1] "t-statistic: 3.2837"
```

```r
print(paste("Degrees of freedom:", df))
```

```
[1] "Degrees of freedom: 12"
```

```r
print(paste("p-value:", round(p_value, 4)))
```

```
[1] "p-value: 0.0065"
```

The `pt()` function in R gives the cumulative distribution function of the t-distribution. The `2 * (1 - pt(abs(t_statistic), df))` line calculates the two-tailed p-value. This will tell us if the slopes are significantly different at your chosen alpha level (commonly 0.05).

12.6 Designed experiments

In the following section, we'll focus on disease data collected over time from the same plot unit, also known as repeated measures. This data comes from a designed experiment aimed at evaluating and comparing the effects of different treatments.

Specifically, we'll use a dataset of progress curves found on page 98 of "Study of Plant Disease Epidemics" (Madden et al. 2007c). These curves depict the incidence of soybean plants

showing symptoms of bud blight, which is caused by the tobacco streak virus. Four different treatments, corresponding to different planting dates, were evaluated using a randomized complete block design with four replicates. Each curve has four time-based assessments.

The data for this study is stored in a CSV file, which we'll load into our environment using the read_csv() function. Once loaded, we'll store the data in a dataframe named budblight.

12.6.1 Loading data

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

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.

```
budblight
```

```
# A tibble: 64 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
 7 PD1      40     3  0.36
 8 PD1      40     4  0.37
 9 PD1      50     1  0.57
10 PD1      50     2  0.52
# i 54 more rows
```

12.6.2 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) +
  scale_color_grey()+
  theme_r4pde()+
  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) +
  scale_color_grey()+
  theme_r4pde()+
  theme(legend.position = "none")+
  labs(y = "Transformed incidence", x = "Time (days after emergence)")
library(patchwork)
p3 | p4
```

12.6.3 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.

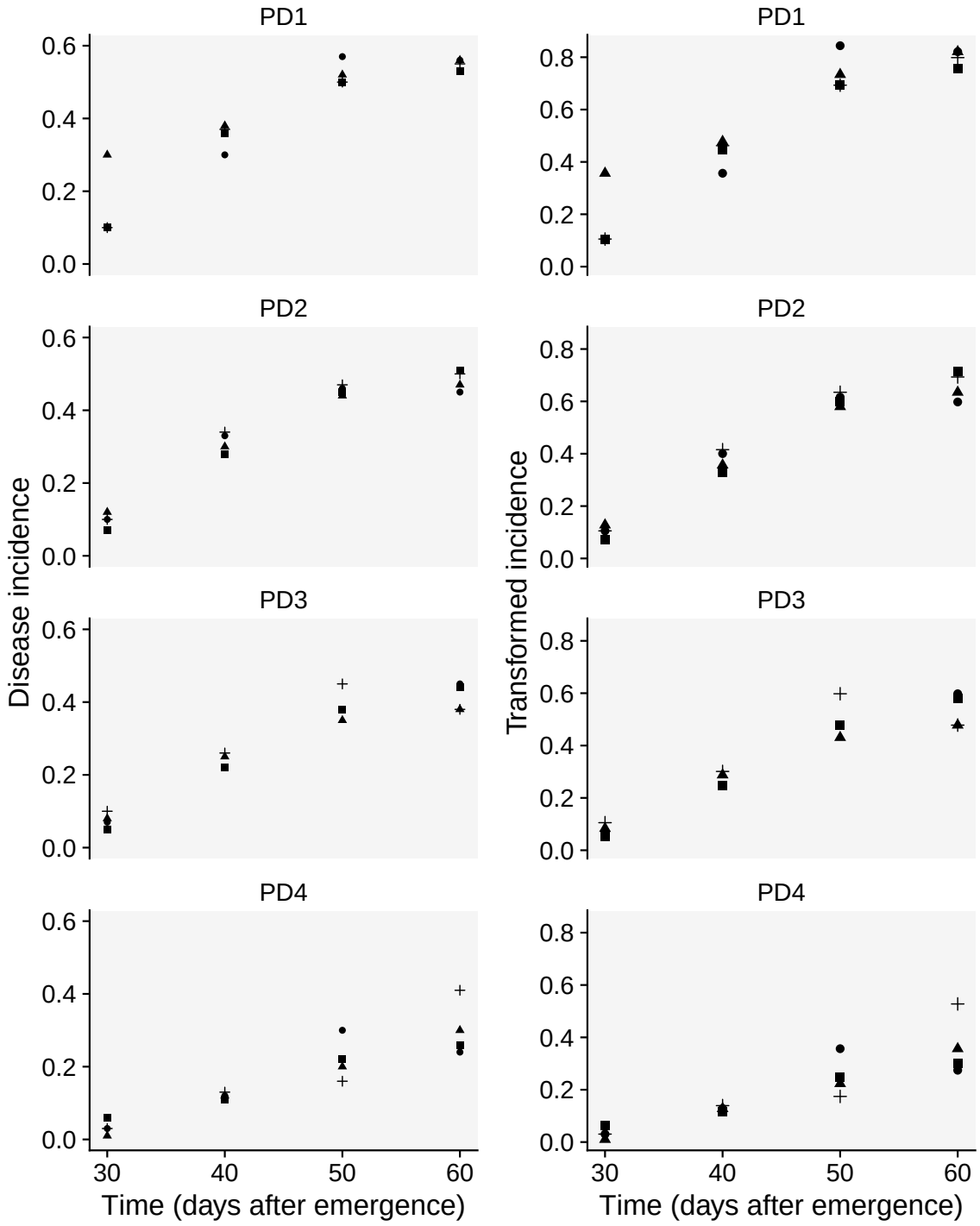


Figure 12.8: Disease progress curves for the incidence of budblight of soybean in Brazil for four planting dates

12.6.3.1 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)) +
  scale_color_grey()+
  theme_r4pde()+
  geom_point(aes(time, y)) +
  geom_line(linewidth = 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)) +
  scale_color_grey()+
  theme_r4pde()+
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = r_ci_lwr, ymax = r_ci_upr),
    width = 0,
    size = 1
  ) +
```

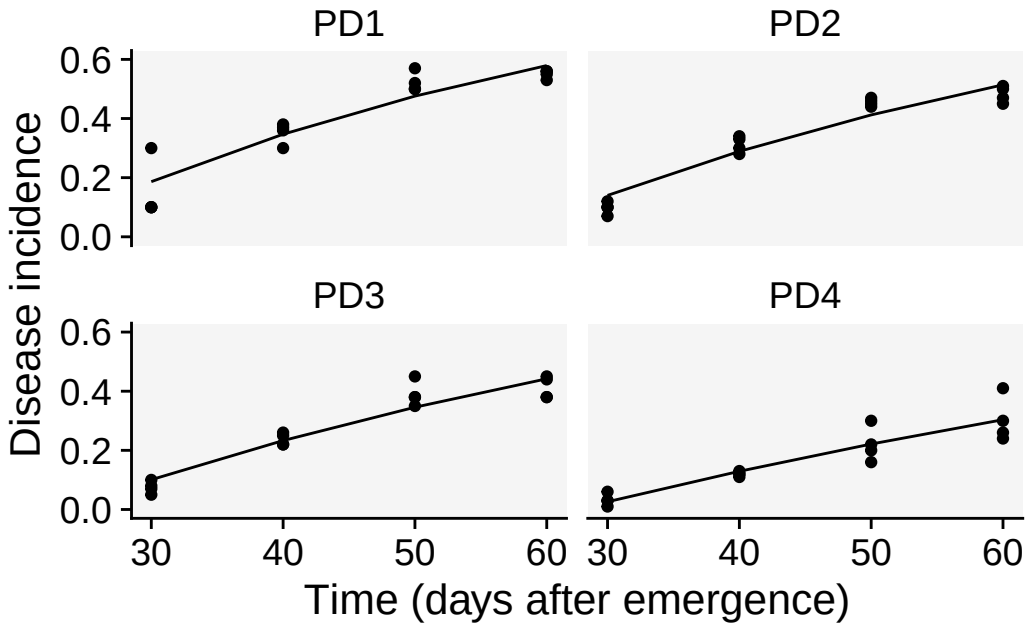


Figure 12.9: Observed (dot) and fitted values by a monomolecular model (line) to the data on the incidence of budblight of soybean in Brazil for four planting dates

```
  labs(
    x = "Epidemic",
    y = "Infection rate (r)"
  )

p6 <- lin1$Parameters |>
  filter(model == "Monomolecular") |>
  ggplot(aes(treat, y0)) +
  scale_color_grey()+
  theme_r4pde()+
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = y0_ci_lwr, ymax = y0_ci_upr),
    width = 0,
    size = 1
  ) +
  labs(
    x = "Time",
    y = "Initial inoculum (y0)"
  )
```

```
p5 | p6
```

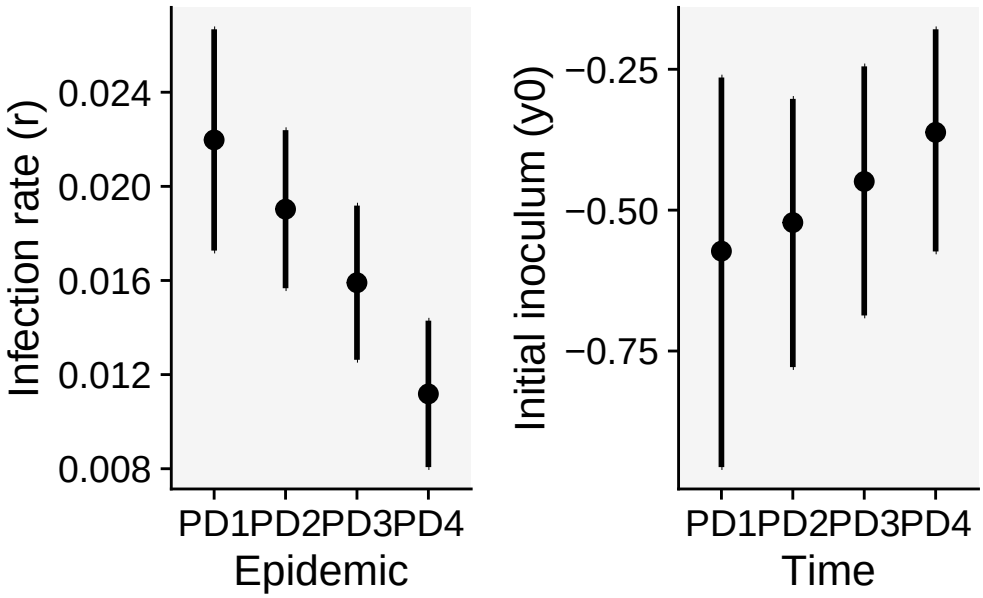


Figure 12.10: Estimates of the infection rate (left) and initial inoculum (right) from the fit of a monomolecular model to the data on the incidence of budblight of soybean in Brazil for four planting dates

12.6.3.2 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
)
```

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
```

```
p7 <- nlin1$Parameters |>
  filter(model == "Monomolecular") |>
  ggplot(aes(treat, r)) +
  scale_color_grey()+
  theme_r4pde()+
  geom_point(size = 3) +
```

```
  geom_errorbar(aes(ymin = r_ci_lwr, ymax = r_ci_upr),
    width = 0,
    size = 1
  ) +
  labs(
    x = "Epidemic",
    y = "Infection rate (r)"
  )

p8 <- nlin1$Parameters |>
  filter(model == "Monomolecular") |>
  ggplot(aes(treat, y0)) +
  scale_color_grey()+
  theme_r4pde()+
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = y0_ci_lwr, ymax = y0_ci_upr),
    width = 0,
    size = 1
  ) +
  labs(
    x = "Epidemic",
    y = "Initial inoculum (y0)"
  )

p7 | p8
```

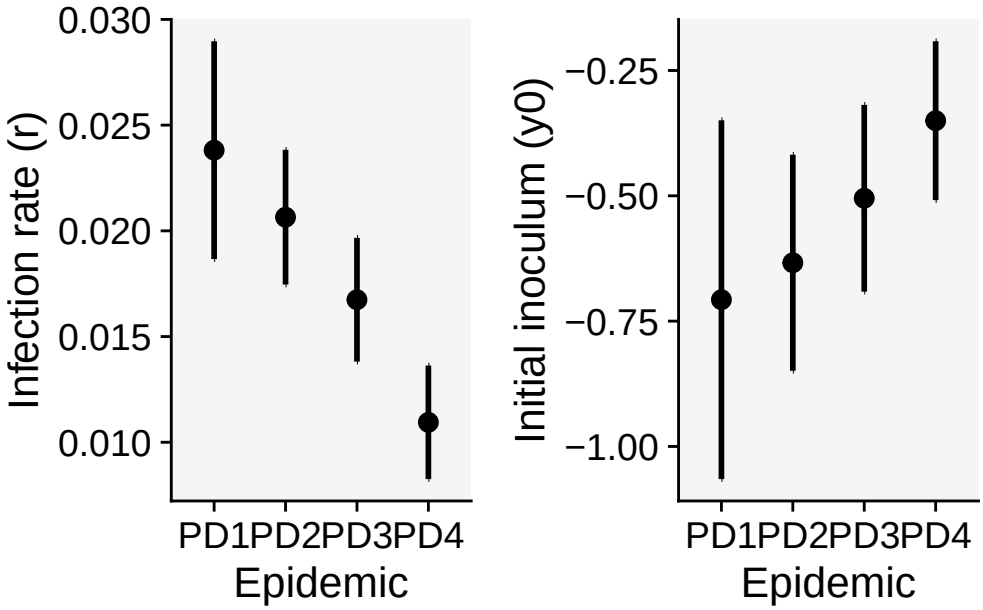


Figure 12.11: Estimates of the infection rate (left) and initial inoculum (right) from the fit of a monomolecular model to the data on the incidence of budblight of soybean in Brazil for four planting dates

Part III

Spatial analysis

13 Spatial gradients

13.1 Introduction

The assessment of disease in terms of its spatial distribution, particularly considering changes in its intensity as it spreads over distance, is defined as the "disease gradient." It's the dispersal, or migration, of the pathogen through various means—such as wind, vectors, rain, movement of infected material, or even human mediation—that encourages the spread of plant diseases within a field or across continents, thereby creating these disease gradients.

There exist two distinct types of gradients: the inoculum gradient, in which the availability of a host is not necessarily a prerequisite, and the disease gradient, where all three elements of the disease triangle are essential.

In the ensuing chapters, we shall explore examples of actual disease gradients measured in the field, each exhibiting its own unique pattern.

Our first example, from Mundt's 1999 study (Mundt et al. 1999), sought to measure the dispersal potential of the pathogenic bacteria, *Xanthomonas oryzae* pv. *oryzae*, which is responsible for leaf blight in rice. This study was conducted 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)
library(r4pde)
theme_set(theme_r4pde())

xo <-
tibble::tribble(
    ~d,   ~y4,   ~y5,
     0, 3.083, 7.185,
  0.22, 0.521,  0.38,
  0.44, 0.083, 0.157,
  0.66, 0.021, 0.028
  )
```

```
g1 <- xo |>
  ggplot(aes(d, y4))+
 theme_r4pde()+
  geom_point(size = 2)+
  geom_line()+
  ylim(0,8)+
  labs(y = "Number of new lesions",
       x = "Distance (m)",
       title = "1994 wet season")

g2 <- xo |>
  ggplot(aes(d, y5))+
  theme_r4pde()+
  geom_point(size = 2)+
  geom_line()+
  ylim(0,8)+
  labs(y = "Number of new lesions",
       x = "Distance (m)",
       title = "1995 wet season")

library(patchwork)
(g1 | g2) +  plot_annotation(
    caption = "Source: Mundt et al. (1999)")
```

The second example of a disease gradient pertains to stripe rust, caused by *Puccinia striiformis* f. sp. *tritici*, on wheat. This data was collected during a field experiment conducted at Hermiston in 2002, as reported in Sackett's 2005 study (Sackett and Mundt 2005). Later, the data was made publicly available in 2015, courtesy of Mikaberidze (Mikaberidze et al. 2015). For our discussion, we'll manually input the data in a tibble format.

This tibble contains five columns. The first and second columns represent distances from the source of infection, denoted in feet and meters, respectively. The remaining three columns consist of measures of stripe rust severity, each from a separate replicated plot. These measurements offer us a quantifiable view of the disease gradient of stripe rust in the field, thereby shedding light on the infection's spatial distribution and intensity.

```
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,
```

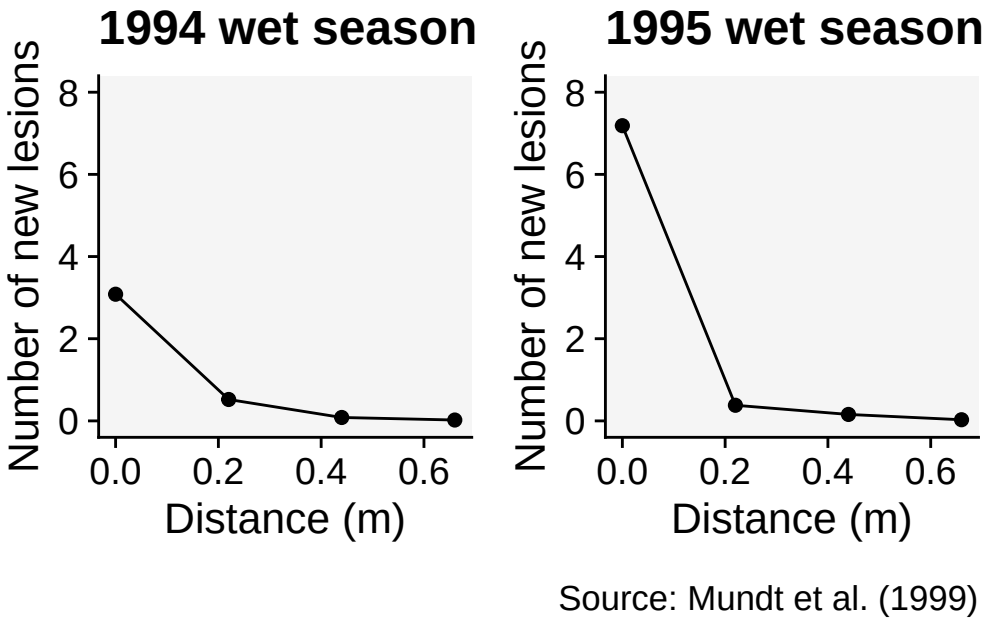


Figure 13.1: Primary gradients of bacerial blight of rice in two wet seasons in the Philippines

```
  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)
```

```r
hermiston |>
  pivot_longer(3:5, names_to = "replicate", values_to = "severity") |>
  ggplot(aes(dist_m, severity, color = replicate))+
  theme_r4pde()+
  theme(legend.position = "bottom")+
  geom_point(size = 2)+
  geom_line(size = 1)+
  scale_color_grey()+
  labs(x = "Distance from the source (m)",
       y = "Stripe rust severity (%)",
       color = "Replicate",
       caption = "source: Sackett et al. (2005)")
```

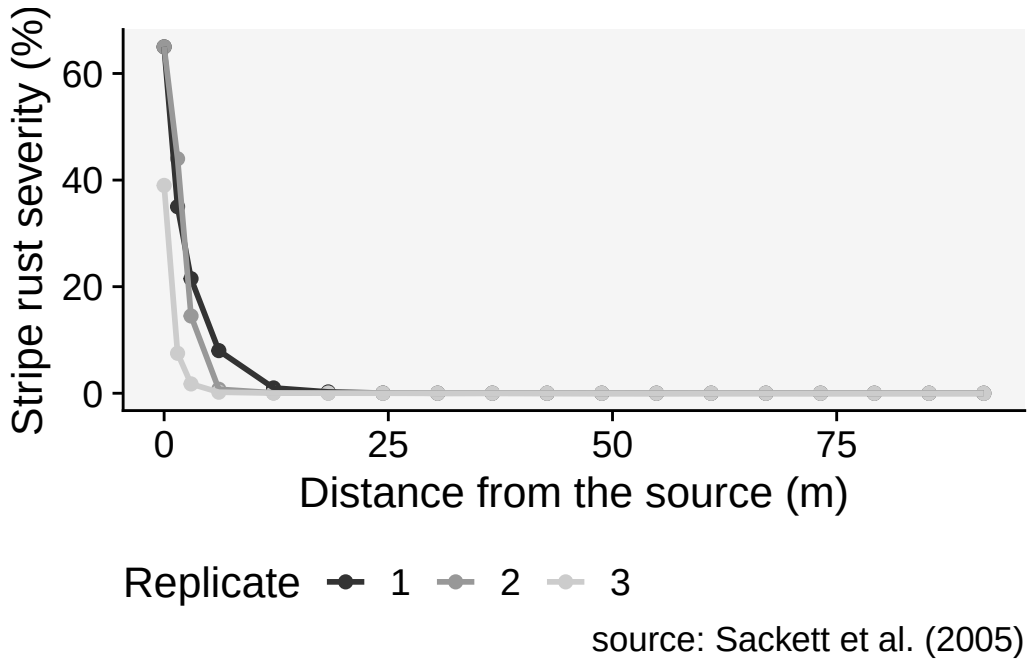


Figure 13.2: Primary gradients of stripe rust of wheat on a replicated experiment

As evidenced by the examples presented above, disease gradients, assuming a single source of inoculum, typically display a pattern wherein the disease's intensity diminishes more steeply within shorter proximities to the source. Conversely, the decrease is less steep at greater distances, eventually reaching a point of either zero or a low background level with only occasional diseased plants.

The unique shapes of these gradients are largely influenced by mechanisms associated with the dispersal of the inoculum, which are contingent not only on the pathogen's biological

characteristics but also heavily upon environmental factors that can impact the pathogen's dispersion.

From this, we can categorize the resulting gradients into two types: primary and secondary. The primary gradient is generated solely from the initial source of the infection. On the other hand, the secondary gradient arises from the movement of inoculum that has been produced by plants previously infected due to the primary gradient. These secondary infections then spread to other plants situated at increasing distances from the initial source.

As the disease proliferates over time, it's expected that a combination of both primary and secondary gradients will manifest. This interplay between the two gradient types contributes to the overall spread and severity of the disease within a given population and environment.

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))+
  theme_r4pde()+
  geom_point()+
  geom_line()+
  scale_color_grey()+
```

```
  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(x = "Distance from focus (m)",
       y = "Percent defoliation",
       color = "Date",
       caption = "Parker et al. (1997)")
```

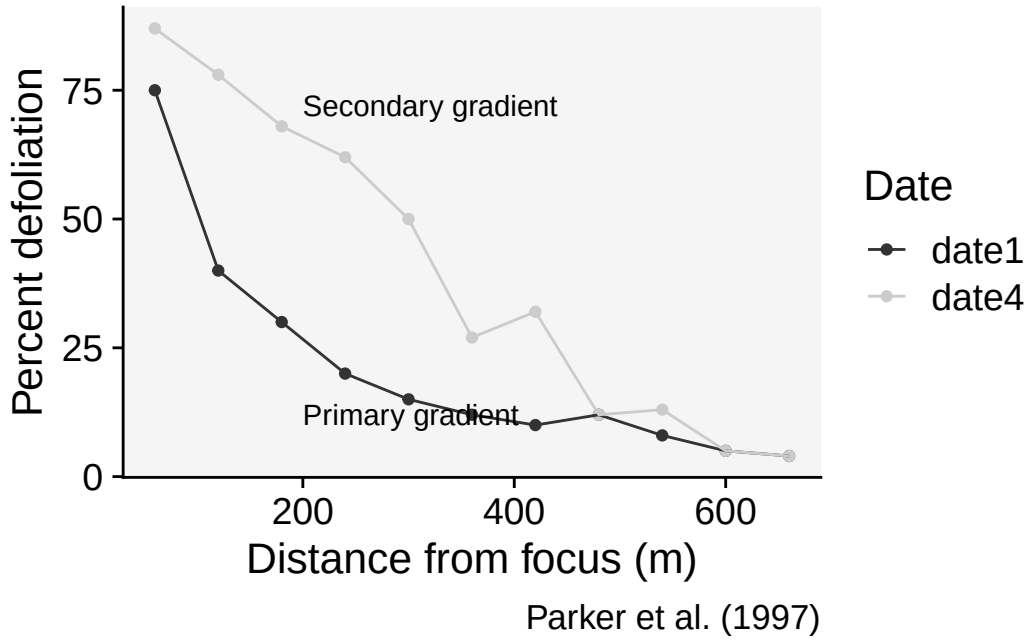


Figure 13.3: Primary and secondary gradients of defoliation due to Septoria leaf spot on tomato

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)

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 = "#3
  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 = "#3
  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 = "#3399
  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")
```

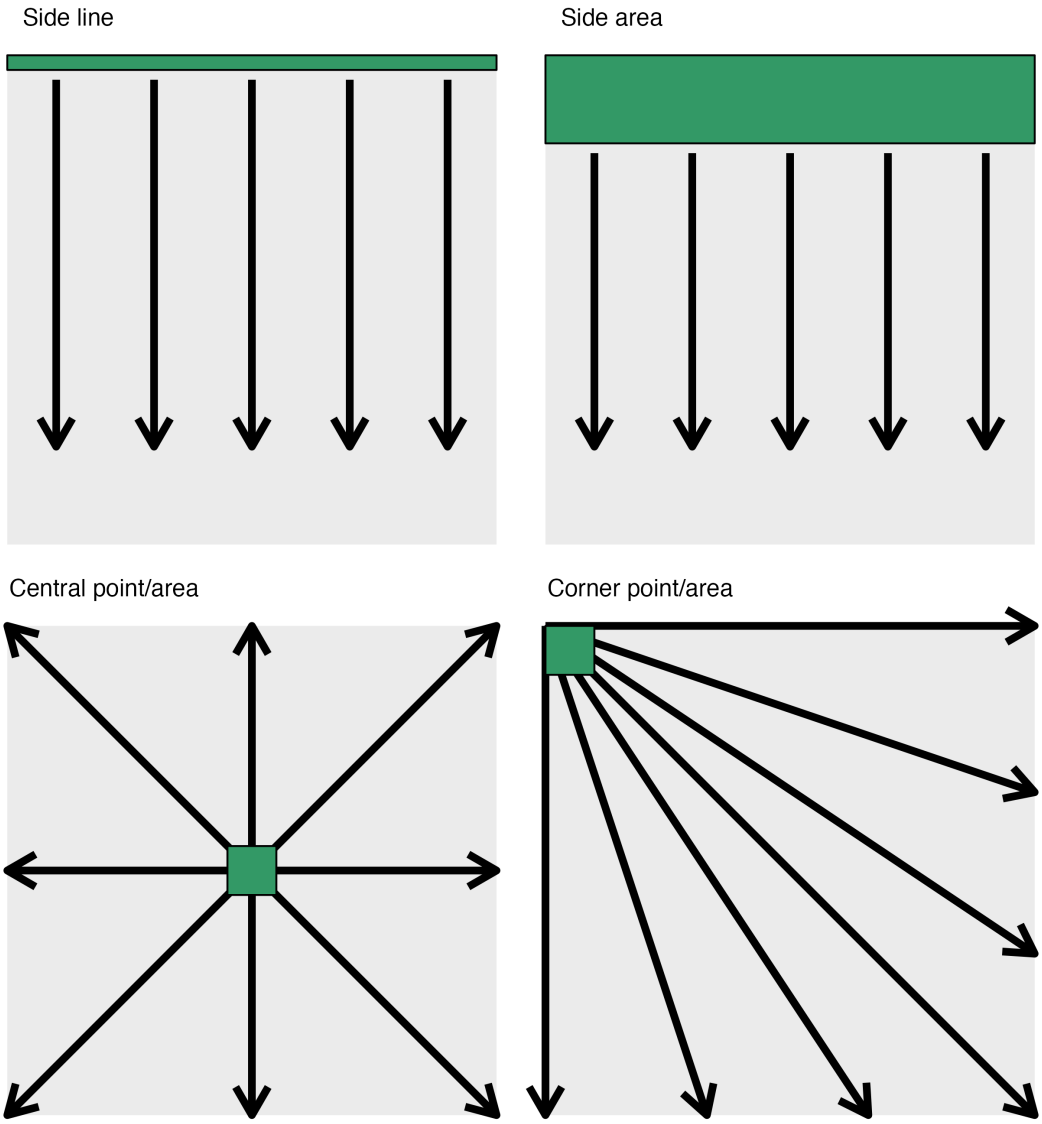


Figure 13.4: Example of location and size of inoculum sources for the study of disease gradients

14 Gradient models

Similar to the 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.

14.1 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(r4pde)
theme_set(theme_r4pde()) # set global theme
```

Set the parameters for the exponential model with two rates and the same inoculum level 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)) +
  theme_r4pde()+
  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(x = "Distance (m)", y = "Disease incidence (proportion)"
  )
```

14.2 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:

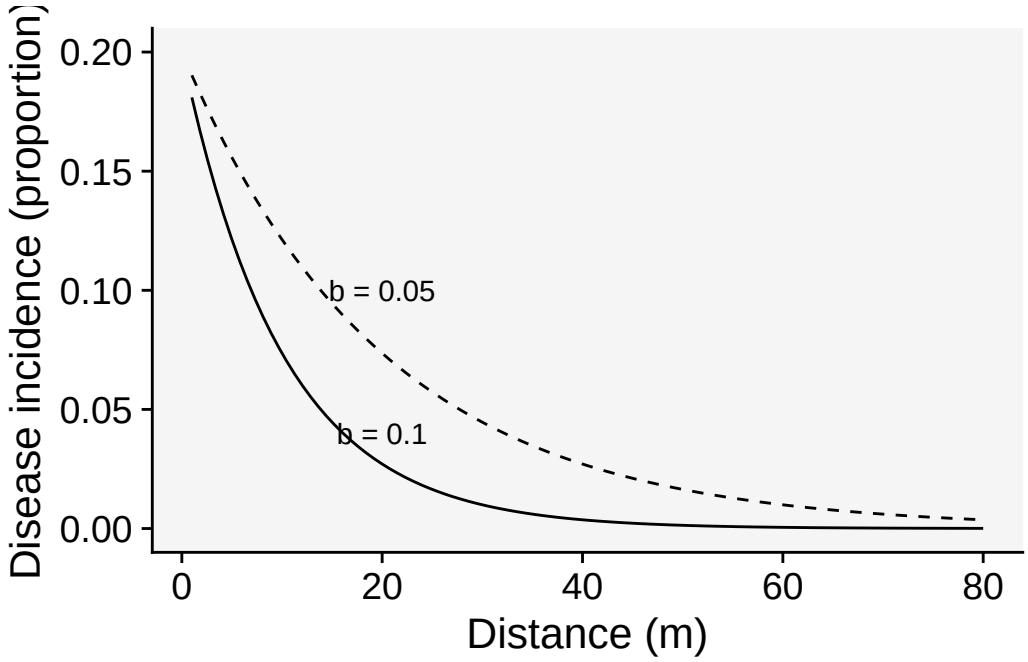


Figure 14.1: Exponential curves describing plant disease gradients

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

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)) +
```

```
theme_r4pde()+
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(x = "Distance (m)", y = "Disease incidence")
```

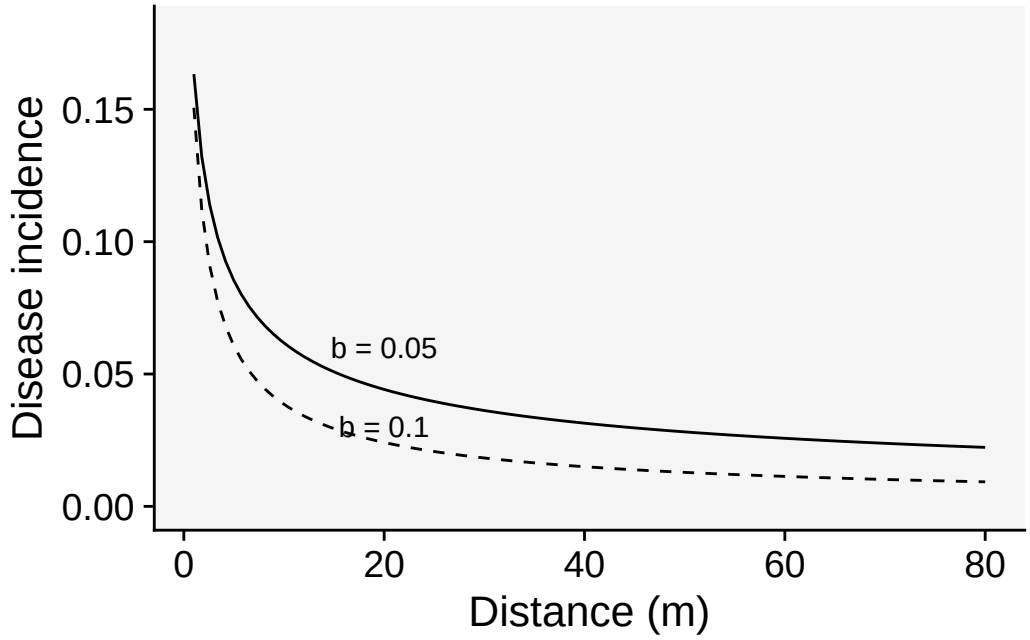


Figure 14.2: Power law (modified) curves describing plant disease gradients

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.

14.3 Linearization of the models

14.3.1 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.

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

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

14.3.2 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 $ln(y)$.

```
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)) +
  theme_r4pde()+
  stat_function(fun = function(x) log(a1) - (b1 * x), linetype = 1) +
  stat_function(fun = function(x) log(a2) - (b2 * x), linetype = 2) +
  labs(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 $ln(x + C)$.

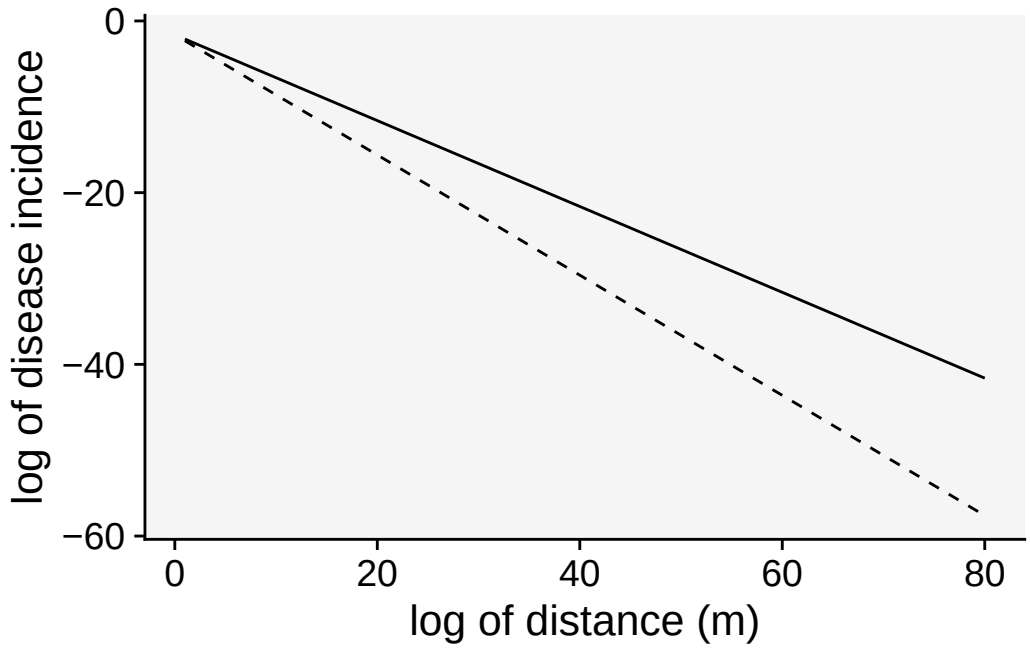


Figure 14.3: Linearization of the exponential model describing plant disease gradients

```
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)) +
  theme_r4pde()+
  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"
  )
```

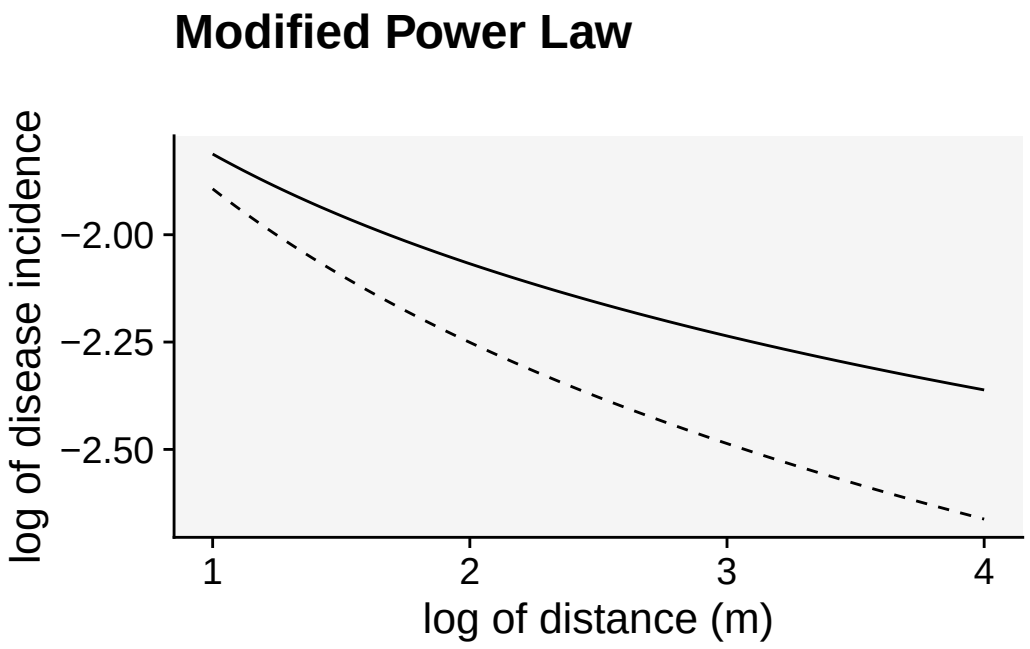


Figure 14.4: Linearization of the modified power law curves describing plant disease gradients

14.4 Interactive application

A shiny app was developed to demonstrate these two models interactively. Click on the image below to get access to the app.

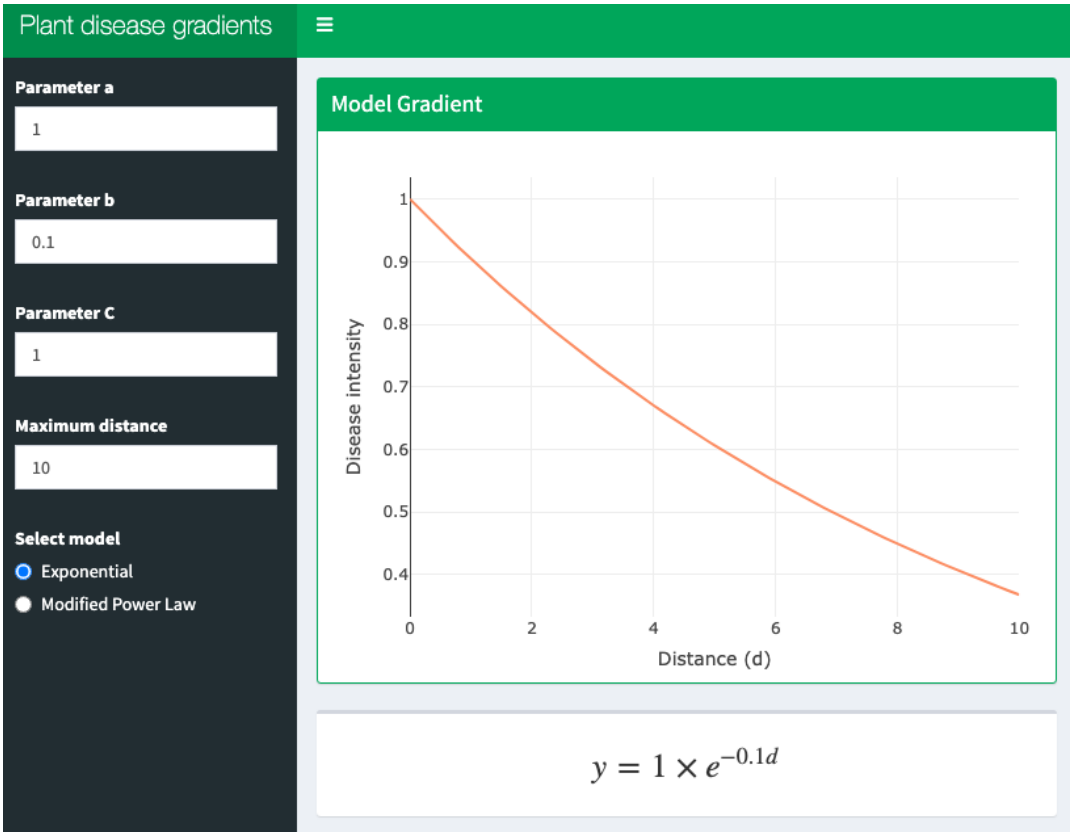


Figure 14.5: Screenshot of the application to visualize the spatial disease gradient models by varying the model's parameters

15 Fitting gradient models

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

```
library(tidyverse)
library(r4pde)
theme_set(theme_r4pde(font_size = 16)) # set global theme
```

15.1 Dataset

The hypothetical data describe the gradient curve 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 data frame 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
knitr::kable(grad1) # show the gradient
```

x	Y
0.8	184.9
1.6	113.3
2.4	113.3
3.2	64.1
4.0	25.0
7.2	8.0
12.0	4.3
15.2	2.5
21.6	1.0
28.8	0.8

15.2 Visualize the gradient curve

The gradient can be visualized using `ggplot` function.

```
grad1 |>
  ggplot(aes(x, Y))+
  theme_r4pde(font_size = 16)+
  geom_point(size = 2)+
  geom_line()+
  labs(y = "Lesion count",
       x = "Distance (m)")
```

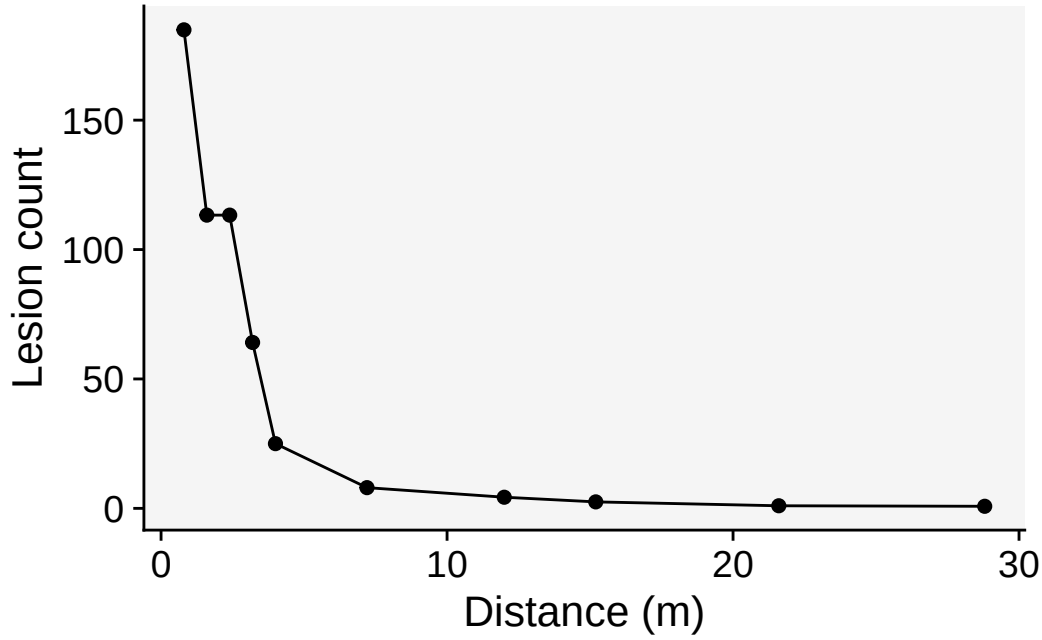


Figure 15.1: Hypothetical gradient of lesion count over distances from the inoculum source

15.3 Linear regression

One method to determine the best-fitting model for gradient data is through linear regression. Depending on the chosen model, the transformed Y variable is regressed against the distance (which could be either in its original form or transformed). By doing this, we can derive the model's parameters and evaluate its fit using various statistics. Two primary ways to appraise the model's fit are by visually inspecting the regression line and examining the coefficient of

determination (often denoted as R^2). Now, let's proceed to fit each of the three models discussed in the previous chapter.

15.3.1 Exponential model

In this model, the log of Y is taken and regressed against the (untransformed) distance from the focus. Let's fit the model and examine the summary output of model fit.

```
reg_exp <- lm(log(Y) ~ x, data = grad1)
jtools::summ(reg_exp)
```

Observations	10
Dependent variable	log(Y)
Type	OLS linear regression

F(1,8)	57.39
R^2	0.88
Adj. R^2	0.86

	Est.	S.E.	t val.	p
(Intercept)	4.58	0.35	13.00	0.00
x	-0.20	0.03	-7.58	0.00

Standard errors: OLS

The intercept a represents the natural logarithm (log) of the response variable when the predictor is at a distance of zero. The negative slope $-b$ indicates the rate at which the response decreases as the predictor increases — this is the decline rate of the gradient. The adjusted R-squared value of 0.86 suggests that approximately 86% of the variability in the response variable can be explained by the predictor in the model. While this seems to indicate a good fit, it is essential to compare this coefficient with those from other models to determine its relative goodness of fit. Furthermore, visually inspecting a regression plot is crucial. By doing this, we can check for any patterns or residuals around the predicted line, which can provide insights into the model's assumptions and potential areas of improvement

```
grad1 |>
  ggplot(aes(x, log(Y)))+
  theme_r4pde(font_size = 16)+
  geom_point(size = 2)+
  geom_line()+
```

```
 geom_abline(slope = coef(reg_exp)[[2]], intercept = coef(reg_exp)[[1]],
             linewidth = 1, linetype = 2)+
labs(y = "Log of Lesion count",
      x = "Distance (m)")
```

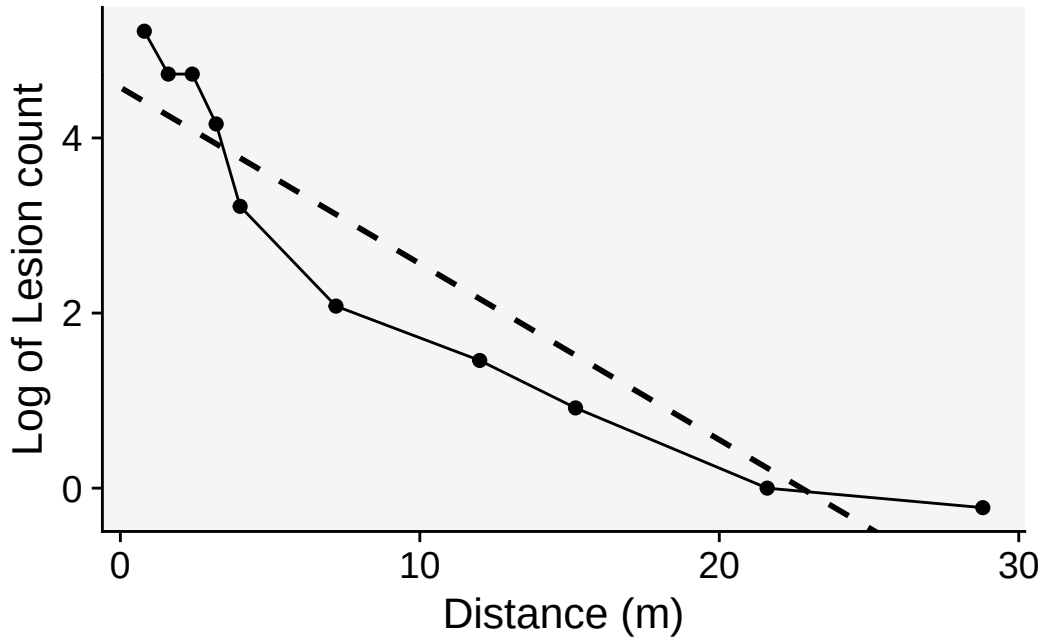


Figure 15.2: Fit of the exponential model to the log of lesion count over distances from the inoculum source

From the aforementioned plot, it's evident that the exponential model might not be the optimal choice. This inference is drawn from the noticeable patterns or residuals surrounding the regression fit line, suggesting that the model may not capture all the underlying structures in the data.

15.3.2 Power law model

For the power law model, we employ a log-log transformation: the natural logarithm (log) of Y is regressed against the log of X. Following this transformation, we apply the regression procedure to determine the model's parameters. Additionally, we extract the relevant statistics to evaluate the model's fit to the data

```
reg_p <- lm(log(Y) ~ log(x), data = grad1)
jtools::summ(reg_p)
```

Observations	10
Dependent variable	log(Y)
Type	OLS linear regression

F(1,8)	203.26
R^2	0.96
Adj. R^2	0.96

	Est.	S.E.	t val.	p
(Intercept)	5.56	0.25	22.66	0.00
log(x)	-1.70	0.12	-14.26	0.00

Standard errors: OLS

The plot presented below underscores the superiority of the power law model in comparison to the exponential model. One of the key indicators of this superior fit is the higher coefficient of determination, R^2 for the power law model. A higher R^2 value suggests that the model can explain a greater proportion of the variance in the dependent variable, making it a better fit for the data at hand.

```
grad1 |>
  ggplot(aes(log(x), log(Y)))+
  theme_r4pde(font_size = 16)+
  geom_point(size = 2)+
  geom_line()+
  geom_abline(slope = coef(reg_p)[[2]], intercept = coef(reg_p)[[1]],
              linewidth = 1, linetype = 2)+
 labs(y = "Log of Lesion count",
       x = "Log of distance")
```

15.3.3 Modified power law model

In the modified power model, a constant is added to x.

```
reg_pm <- lm(log(Y) ~ log(x + 0.4), data = grad1)
jtools::summ(reg_pm)
```

```
grad1 |>
  ggplot(aes(log(x+0.4), log(Y)))+
```

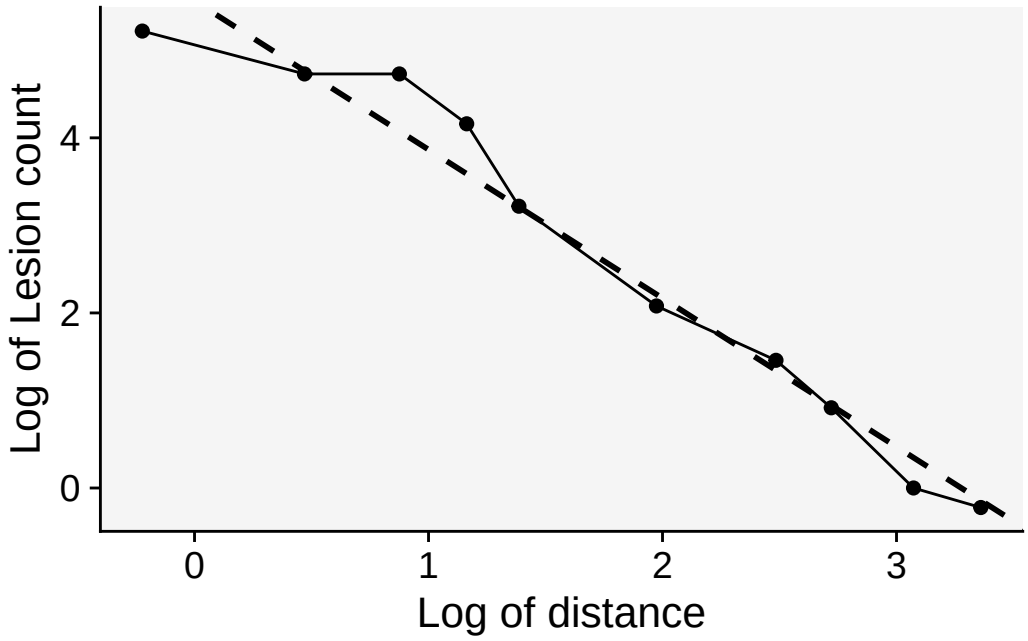


Figure 15.3: Fit of the power law model to the log of lesion count over log of the distance from the inoculum source

Observations	10
Dependent variable	log(Y)
Type	OLS linear regression

F(1,8)	302.16
R^2	0.97
Adj. R^2	0.97

	Est.	S.E.	t val.	p
(Intercept)	6.10	0.23	26.73	0.00
log(x + 0.4)	-1.88	0.11	-17.38	0.00

Standard errors: OLS

```
  theme_r4pde(font_size = 16)+
  geom_point(size = 2)+
  geom_line()+
  geom_abline(slope = coef(reg_pm)[[2]], intercept = coef(reg_pm)[[1]],
```

```
            linewidth = 1, linetype = 2)+
labs(y = "Log of Lesion count",
      x = "Log of distance + 0.4 (m)")
```

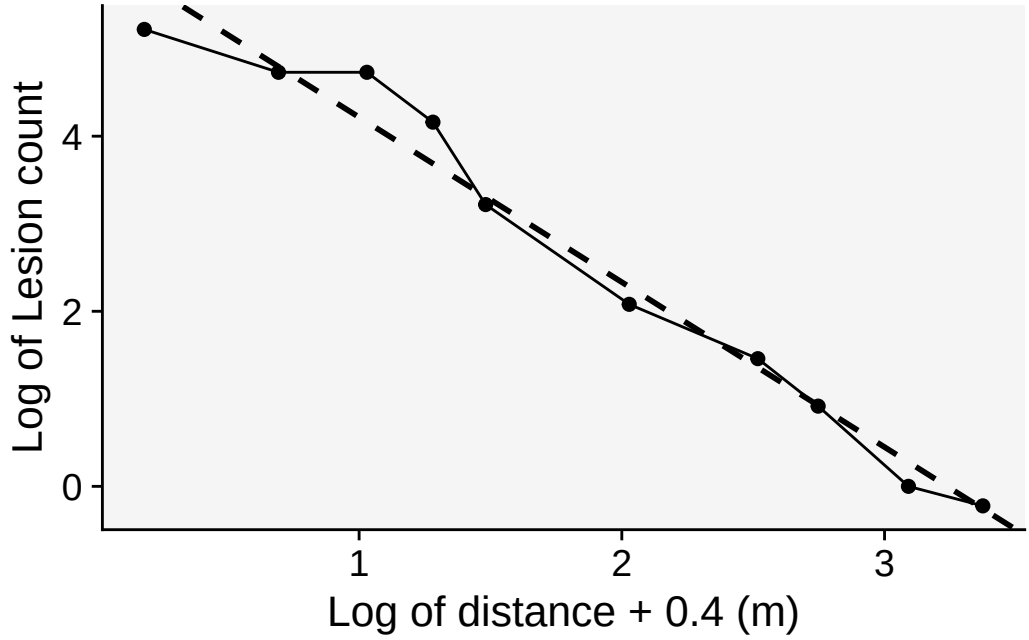
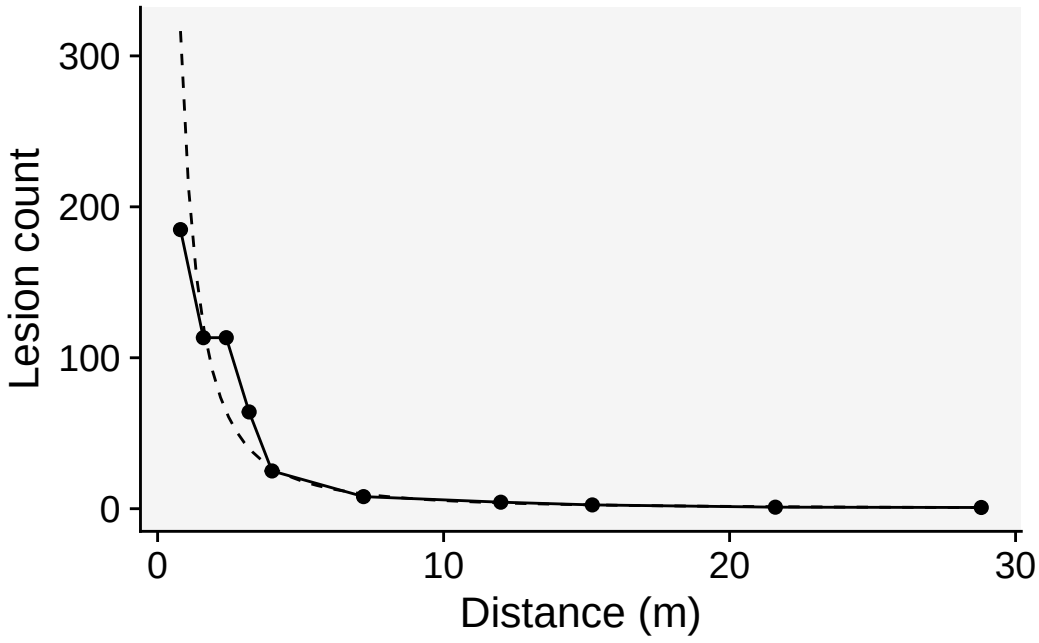


Figure 15.4: Fit of the modified power law model to the log of lesion count over log + 0.4 of the distances from the inoculum source

Among the models tested, the modified power law emerges as the most suitable choice based on its highest coefficient of determination, R^2 . This conclusion is not only supported by the statistical metrics but also visibly evident when we examine the graphs of the fitted models.To further illustrate this, we'll generate a gradient plot. On this plot, we'll overlay the data with the best-fitting model — the modified power law. Remember, to accurately represent the data, we'll need to back-transform the parameter a before plotting.

```
grad1 |>
  ggplot(aes(x, Y))+
  theme_r4pde(font_size = 16)+
  geom_point(size = 2)+
  geom_line()+
  stat_function(fun = function(x) intercept = exp(coef(reg_pm)[[1]]) * ((x + 0.4)^coef(reg
  labs(y = "Lesion count",
       x = "Distance (m)")
```



15.4 fit_gradients

The `fit_gradients()` function of the {r4pde} package is designed to take in a dataset consisting of two variables: distance (x) and some measure of the phenomenon (Y). Using this data, the function fits each of the three models and evaluates their performance by calculating the R-squared value for each fit. The higher the R-squared value, the better that particular model explains the variation in the data. Once the models are fit, the function returns a series of outputs:

- A table that summarizes the parameters and fit statistics of each model.
- Diagnostic plots that show how well each model fits the data in its transformed space.
- Plots that juxtapose the original, untransformed data against the fits from each of the three models.

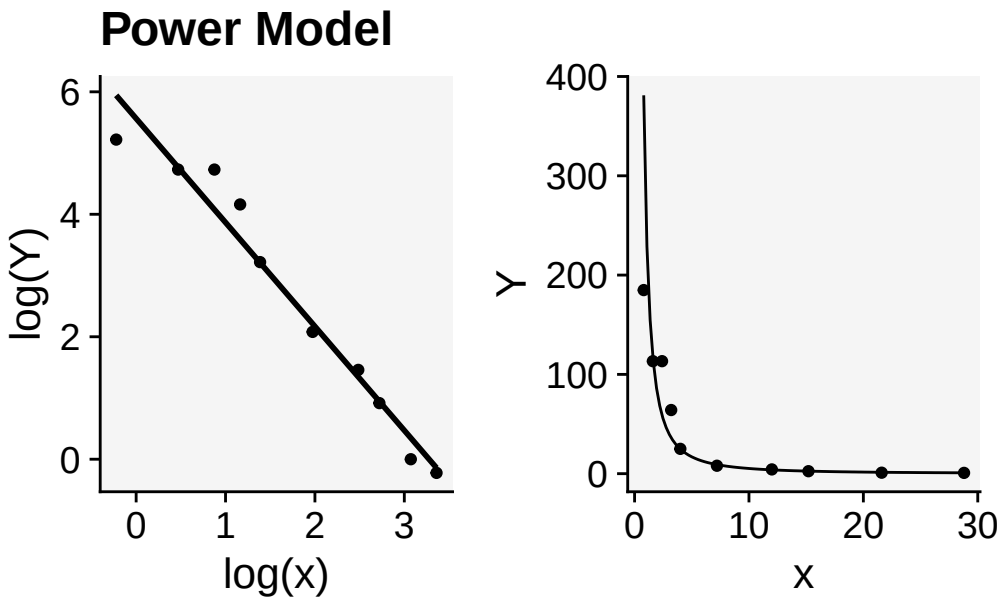
A notable feature is the addition of a constant (C) that can be adjusted in the modified power model. This provides flexibility in tweaking the model to better fit the data if necessary. By providing a comparative analysis of three gradient models, it enables users to quickly identify which model best represents the spatial patterns in their data.

Here is how to use the function with our `grad1` dataset. Then we show the table and two plots as outputs.

```r
library(r4pde)
theme_set(theme_r4pde(font_size = 16))

fit1 <- fit_gradients(grad1, C = 0.4)

knitr::kable(fit1$results_table) # display the table with coefficients and stats
```

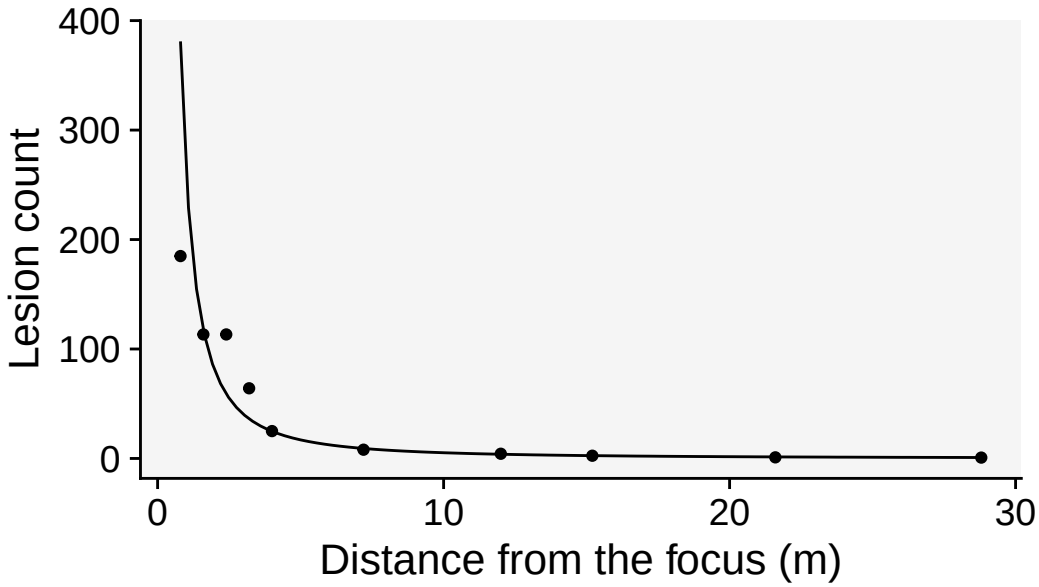
	a.(Intercept)	se_a	sig_a	b.x	se_b	sig_b	a_back.(Intercept)	R2
Exponential	4.577	0.352	**	-0.201	0.027	**	97.222	0.878
Power	5.564	0.246	**	-1.698	0.119	**	260.864	0.962
Modified_Power	6.101	0.228	**	-1.884	0.108	**	446.304	0.974

```r
library(patchwork) # to place plots side by side
(fit1$plot_power  |
  fit1$plot_power_original)+
  labs(title = "")
```



Each plot can be further customized for publication purposes.

```
fit1$plot_power_original +
 labs(x = "Distance from the focus (m)",
      y = "Lesion count",
      title = "")
```



16 Spatial patterns

16.1 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. 2007b). 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 units 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
dat <- data.frame(x,y) # dataframe for plotting
```

Now, the plot to visualize the random pattern.

```r
library(tidyverse)
library(r4pde)
theme_set(theme_r4pde())

pr <- dat |> # R base pipe operator
  ggplot(aes(x, y))+
  theme_r4pde(font_size = 12)+
  geom_point(size =3,
             color = "darkred")+
  ylim(0,30)+
  xlim(0,30)+
  coord_fixed()+
  labs(x = "Distance x", y = "Distance y",
       title = "Random")
pr
```

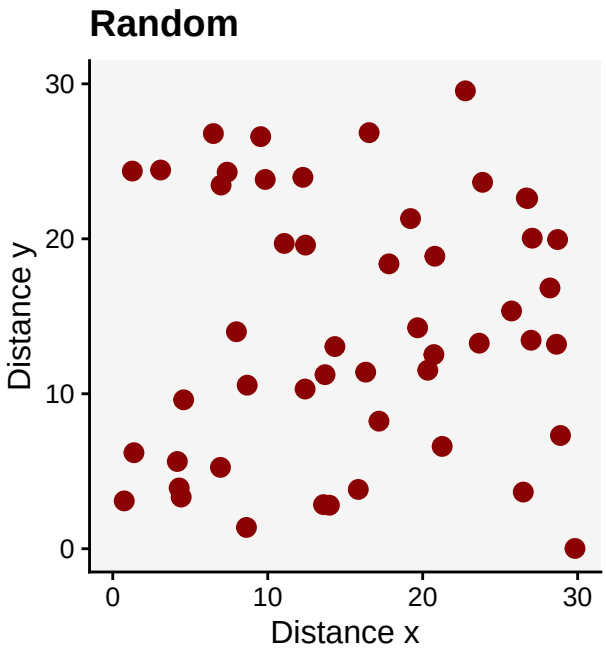


Figure 16.1: Random pattern of a plant disease epidemic

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))+
  theme_r4pde(font_size = 12)+
  geom_point(size = 3, color = "darkred")+
  ylim(0,30)+
  xlim(0,30)+
  coord_fixed()+
  labs(x = "Distance x", y = "Distance y",
       title = "Aggregated")
pag
```

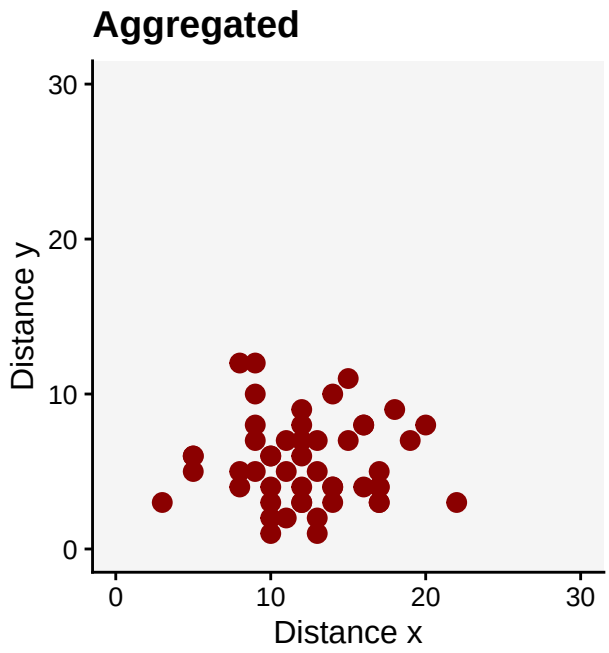


Figure 16.2: Aggregated pattern of a plant disease epidemic

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))+
  theme_r4pde(font_size = 12)+
  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()`).
```

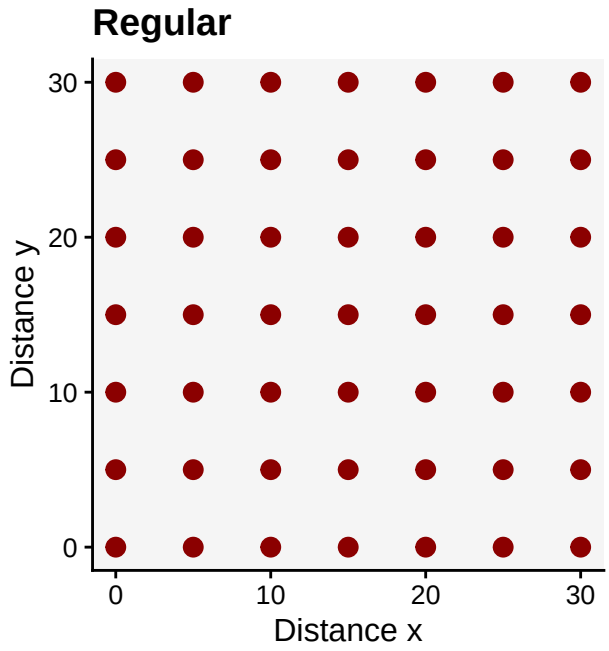


Figure 16.3: Regular pattern of a plant disease epidemic

```
library(patchwork)
preg + pr + pag
```

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

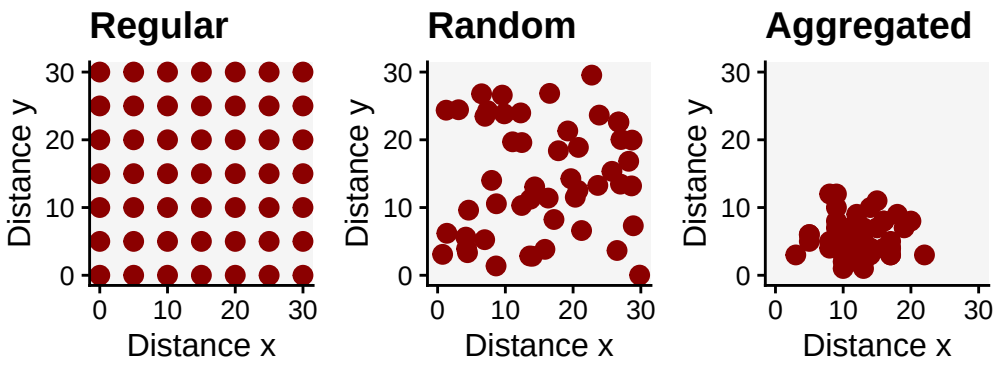


Figure 16.4: Patterns of a plant disease epidemic

16.2 Spatiotemporal

The location of diseased plants can be assessed over time and so we can appraise both the progress and pattern of the epidemics. Let's visualize spatial data collected from actual epidemics monitored (plant is diseased or not diseased) during six times during the epidemics. The data is available in the *epiphy* R package. Let's use only one variety and one irrigation type.

```
library(epiphy)
tswv_1928 <- tomato_tswv$field_1928

tswv_1928 |>
  filter(variety == "Burwood-Prize"&
         irrigation == "trenches") |>
  ggplot(aes(x, y, fill= factor(i)))+
  geom_tile(color = "black")+
```

```
  coord_fixed()+
  scale_fill_manual(values = c("grey70", "darkred"))+
  labs(fill = "Status", title = "")+
  theme_void()+
  theme(legend.position = "bottom")+
  facet_wrap(~ t, nrow =1)
```

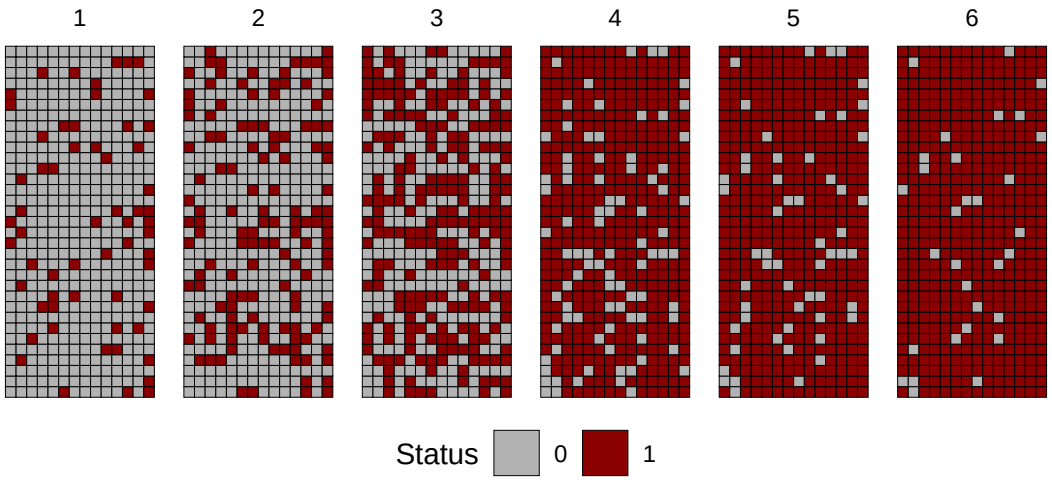


Figure 16.5: Spatial patterns of tomato spotted wilt virus at six assessment times

In this other example, the severity of gummy stebem blight (*Didymella bryoniae)* of watermelon (Café-Filho et al. 2010) was recorded in a 0-4 ordinal scale over time (days after planting) and space, in a naturally-infected rain-fed commercial field, to evaluate the effect of the distance of initial inoculum on the intensity of the disease. The dataset is included in the {r4pde} package that accompanies the book.

```
library(r4pde)
df <- DidymellaWatermellon

df |>
  ggplot(aes(NS_col, EW_row, fill = severity))+
  coord_fixed()+
  geom_tile (color = "white")+
```

```
 theme_void()+
 theme(legend.position = c(0.9,0.25))+
scale_fill_gradient(low = "grey70", high = "darkred")+
 facet_wrap(~ dap, ncol = 4)
```

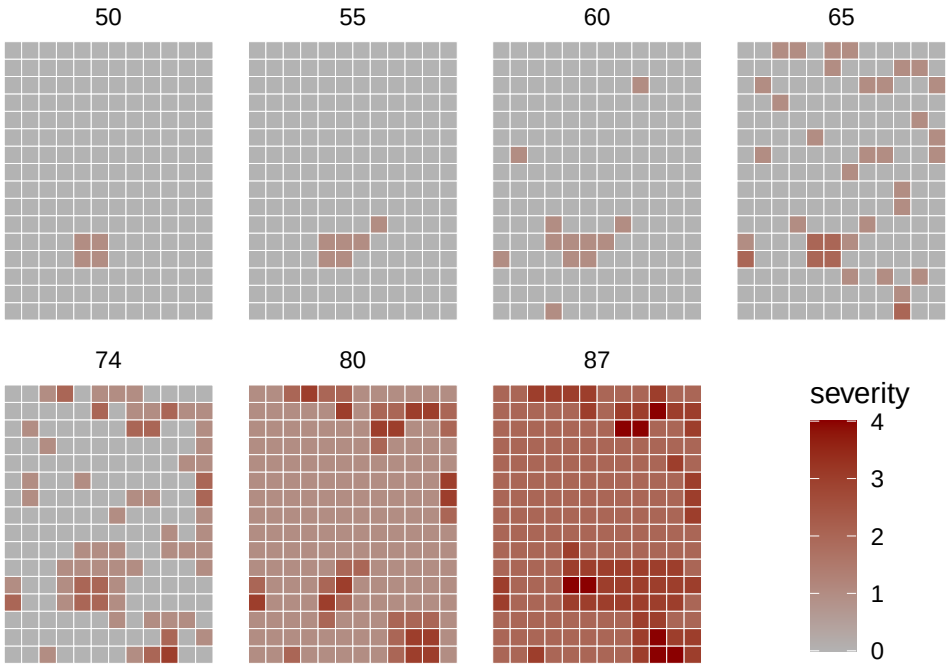


Figure 16.6: Spatial patterns of the severity (0-4 ordinal scale) gummy stem blight of watermelon during seven times after the day of planting (Café-Filho et al. 2010)

We can see that the disease spread from the initial focus detected at 50 days after planting, taking the entire field 37 days later with various levels of severity.

16.3 Simulating spatial patterns

Two Shiny apps have been developed to allow simulating various spatial disease patterns. The first generates a disease- or pathogen-only data where the units are located in a scatter plot where the user can define the number of cells of the grid as well as the number of points to be plotted and the realized pattern: random or aggregated.

The second app generates an artificial plantation with presence-absence data in a 2D map. The user can define the number of rows and number of plants per row and the realized pattern: random or aggregated. The latter pattern can start from the center or border of the plantation. The app calculates the number of foci and the final incidence (proportion of diseased plants).

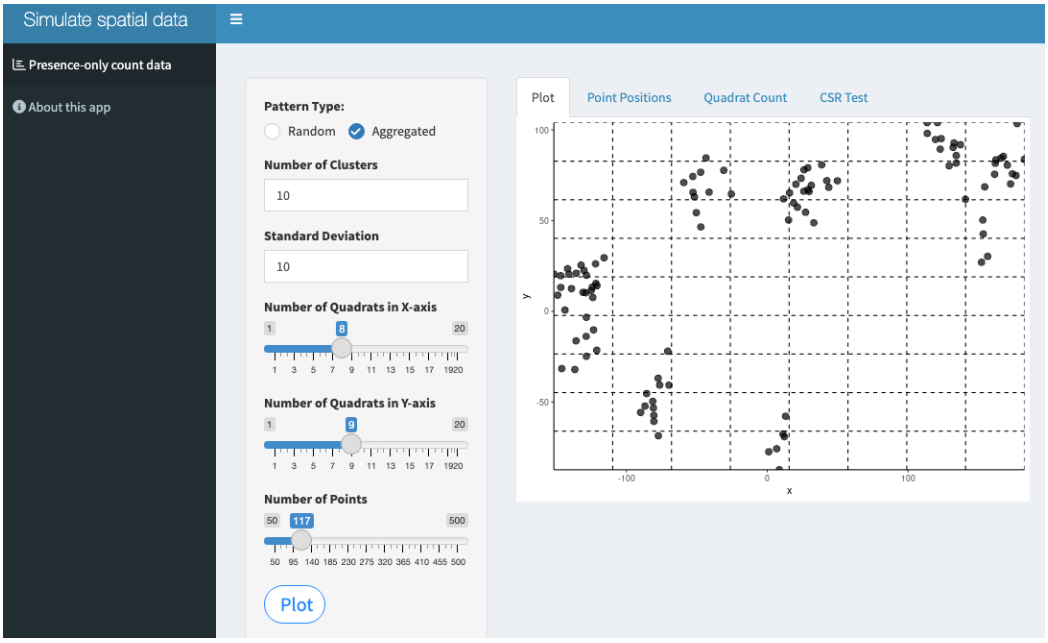


Figure 16.7: Screenshot of a Shiny app to simulate disease-only data in a grid

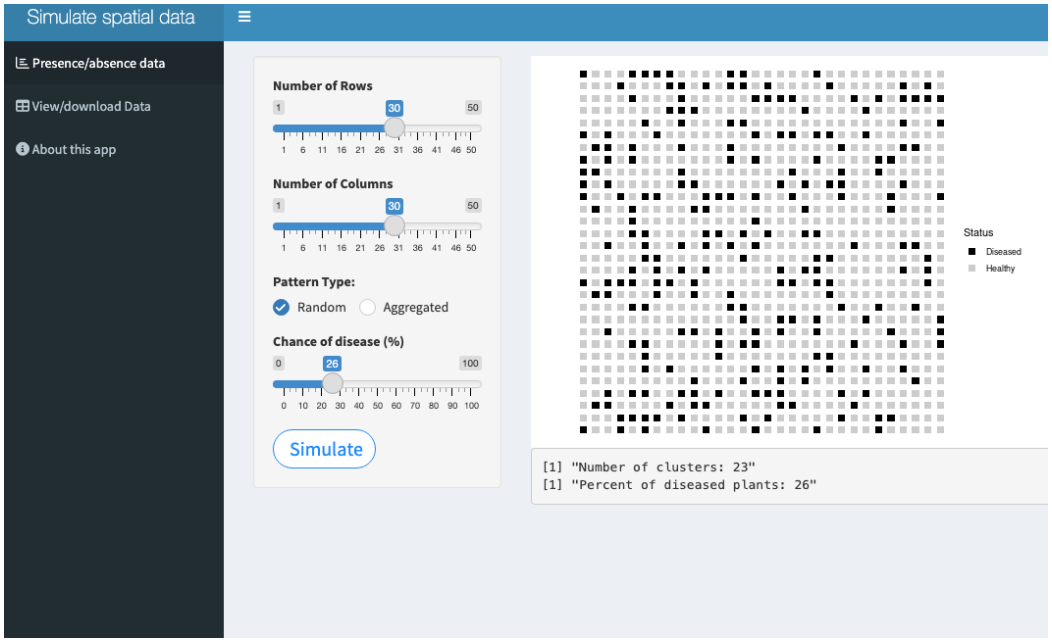


Figure 16.8: Screenshot of a Shiny app to simulate a presence-absence data in a 2D map

17 Tests for patterns

A range of techniques, most based on statistical tests, can be used to detect deviations from randomness in space. The choice of the methods depends on the question, the nature of the data and scale of observation. Usually, more than one test is applied for the same or different scales of interest depending on how the data are collected.

The several exploratory or inferential methods can be classified based on the spatial scale and type of data (binary, count, etc.) collected, but mainly if the spatial location of the unit is known (mapped) or not known (sampled). Following Madden et al. (2007d), two major groups can be formed. The first group uses intensively mapped data for which the location [x,y] of the sampling unit is known. Examples of data include binary data (plant is infected or not infected) in planting row, point pattern (spatial arrangements of points in a 2-D space) and quadrat data (grids are superimposed on point pattern data). The second group is the sparsely sampled data in the form of count or proportion (incidence) data for which the location is not known or, if known, not taken into account in the analysis.

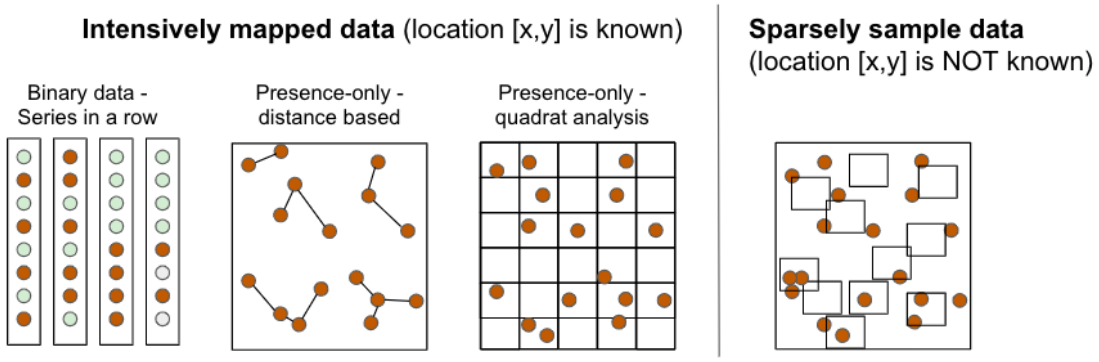


Figure 17.1: Classification of spatial data and methods used to study spatial patterns of plant disease based on knowledge of the location of the sampling units and nature of the data.

17.1 Intensively mapped

17.1.1 Binary data

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

17.1.1.1 Runs test

A **run** is defined as a succession of *one or more* diseased (D) or healthy (H) plants, which are followed and preceded by a plant of the other disease status or no plant at all. In the example below, we can count 13 runs.

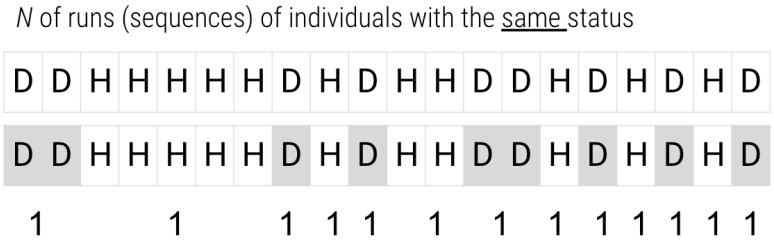


Figure 17.2: Example for the computation of the number of ordinary runs in a sequence of binary data

There would be few runs if there is an aggregation of diseased or healthy plants and a large number of runs for a random mixing of diseased and healthy plants.

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

```
library(tidyverse)
theme_set(theme_bw(base_size = 16))
```

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

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

```
row1 |>
  ggplot(aes(x1, z1, label = x1, color = factor(y1))) +
  geom_point(shape = 15, size = 7) +
  theme_void() +
  scale_x_continuous(breaks = max(z1)) +
  scale_color_manual(values = c("gray70", "darkred")) +
  geom_text(vjust = 0, nudge_y = 0.5) +
  coord_fixed() +
  ylim(0, 2.5) +
  theme(legend.position = "top") +
  labs(color = "Status")
```

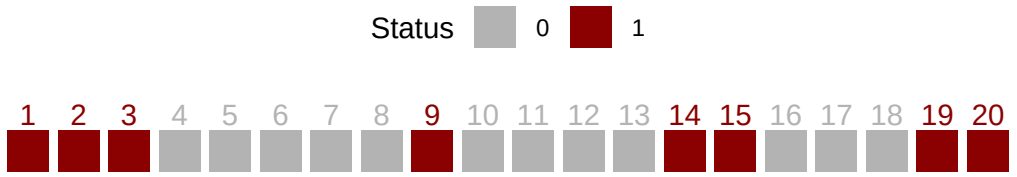


Figure 17.3: Sequence of diseased (1) or non-diseased (0) units (plants). The numbers represent the position of the unit

We can obtain the number of runs and related statistics using the `oruns.test()` function of the *r4pde* package.

```
library(r4pde)
oruns.test(row1$y1)
```

```
$U
[1] 7

$EU
[1] 10.6

$Z
[1] -1.727008

$pvalue
[1] 0.08416615

$result
[1] "clustering"
```

17.1.1.2 Doublets

Doublet analysis is used to compare the observed number or adjacent diseased plants, a doublet (DD or 11), to the number expected if the disease were randomly distributed in the field. If the observed number is greater than the expected number, contagion within the field is suspected. The example below shows 8 doublets.

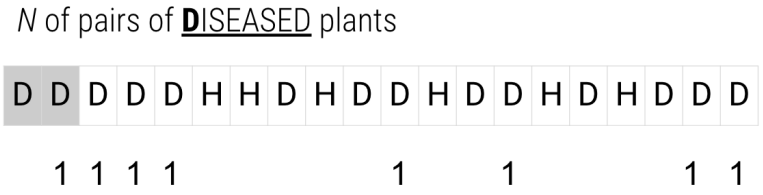


Figure 17.4: Example for the computation of the number of doublets (DD) in a sequence of binary data

The `doublets.test()` function of the *r4pde* package calculates the doublets and associated statistics.

```
doublets.test(row1$y1)
```

```
$Db
```

```
[1] 4

$EDb
[1] 2.8

$ZDb
[1] 0.7559289

$pvalue
[1] 0.4496918

$result
[1] "randomness"
```

17.1.1.3 Join count

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

In R, we can use the `join.count()` function of the {spdep} package to perform a joint count test. First, we need to create the series of binary data from top to bottom and left to right. The data are shown in Fig. 9.13 in page 260 of the book chapter on spatial analysis (Madden et al. 2007b). In the example, there are 5 rows and 5 columns. This will be informed later to run the test.

```
S2 <- c(1,0,1,1,0,
        1,1,0,0,0,
        1,0,1,0,0,
        1,0,0,1,0,
        0,1,0,1,1)
```

Visualize the two-dimensional array:

```
# Convert to raster
mapS2 <- terra::rast(matrix(S2, 5 , 5))
# Convert to data frame
mapS3 <- terra::as.data.frame(mapS2, xy = TRUE)
mapS3 |>
```

```
  ggplot(aes(x, y, label = lyr.1, fill = factor(lyr.1))) +
  geom_tile(color = "white", size = 0.5) +
  theme_void() +
  coord_fixed()+
  labs(fill = "Status") +
  scale_fill_manual(values = c("gray70", "darkred"))+
  theme(legend.position = "top")
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```

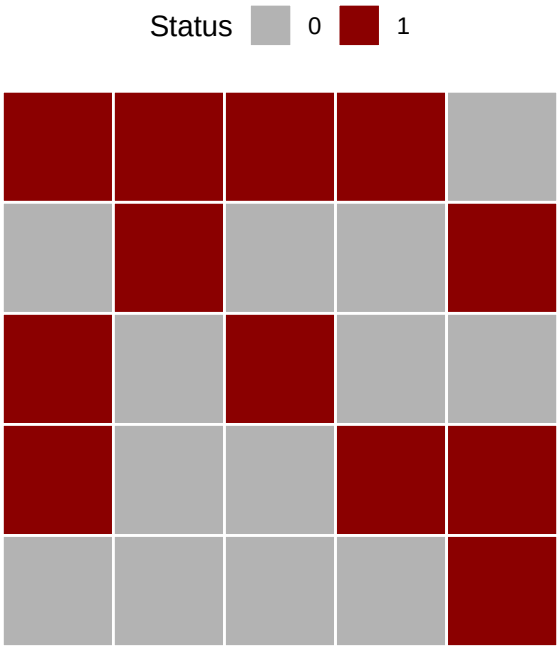


Figure 17.5: Visualization of a matrix of presence/absence data representing a disease spatial pattern

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

```
library(spdep)
nb <- cell2nb(nrow = 5,
```

```
                ncol = 5,
                type = "rook")
```

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

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

```
    Join count test under nonfree sampling

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

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


    Join count test under nonfree sampling

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

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

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

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

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

```
$U
[1] 17

$EU
[1] 13.48

$Z
[1] 1.440688

$pvalue
[1] 0.1496727

$result
[1] "clustering"
```

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

```
$Db
[1] 3

$EDb
[1] 5.28

$ZDb
[1] -1.034484

$pvalue
[1] 0.3009097

$result
[1] "randomness"
```

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

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

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

```
    Join count test under nonfree sampling

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

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


    Join count test under nonfree sampling

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

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

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

```
$U
[1] 8

$EU
[1] 13.48
```

```
$Z
[1] -2.24289

$pvalue
[1] 0.02490392

$result
[1] "clustering"
```

We can apply these tests for a real example epidemic data provided by the epiphy R package (Gigot 2018). Let's work with part of the intensively mapped data on the incidence of tomato spotted wilt virus (TSWV) disease in field trials reported by Cochran (1936) and Bald (1937). First, we need to load the library and then assign one dataframe (the dataset has two dataframes) of the dataset `tomato_tswv` to a new dataframe called `tswv_1929`.

```
library(epiphy)
tswv_1929 <- tomato_tswv$field_1929
tswv_1929 |>  head(10)
```

```
   x  y t i n
1  1  1 1 0 1
2  1  2 1 1 1
3  1  3 1 0 1
4  1  4 1 1 1
5  1  5 1 0 1
6  1  6 1 0 1
7  1  7 1 0 1
8  1  8 1 0 1
9  1  9 1 1 1
10 1 10 1 0 1
```

The inspection of the first 10 rows of the dataframe shows five variables where x and y are spatial grid coordinates, t is assessment time, i is the status of the plant (0 = healthy, 1 = diseased) and n is the sampling unit size (here all one). Let's visualize these data for each sampling time.

```
tswv_1929 |>
  ggplot(aes(x, y, fill = factor(i))) +
  geom_tile() +
  coord_fixed() +
```

```
  scale_fill_manual(values = c("gray70", "darkred")) +
  facet_wrap( ~ t) +
  labs(fill = "Status")+
  theme_r4pde(font_size =12)+
  theme(legend.position = "top")
```

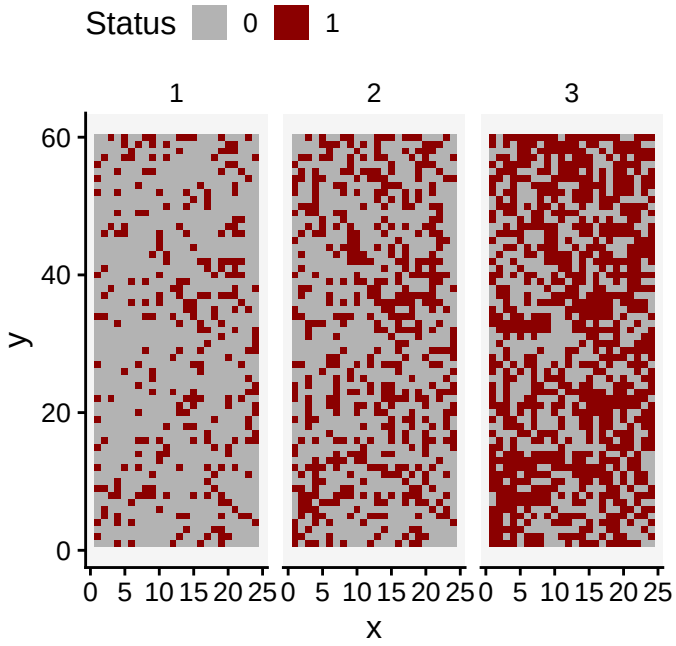


Figure 17.6: Incidence maps for for tomato spotted wilt virus (TSWV) disease in field trials reported by Cochran (1936) and Bald (1937)

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

```
tswv_1929 |>
  dplyr::select(x, y) |>
  summary()
```

```
       x                y         
 Min.   : 1.00   Min.   : 1.00  
 1st Qu.: 6.75   1st Qu.:15.75  
 Median :12.50   Median :30.50  
 Mean   :12.50   Mean   :30.50  
```

```
 3rd Qu.:18.25   3rd Qu.:45.25
 Max.   :24.00   Max.   :60.00
```

There are 60 rows and 24 columns.

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

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

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

```
    Join count test under nonfree sampling

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

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


    Join count test under nonfree sampling

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

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

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

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

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

```
    Join count test under nonfree sampling

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

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


    Join count test under nonfree sampling

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

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

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

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

```
                nb2listw(nb1))
```

```
    Join count test under nonfree sampling

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

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


    Join count test under nonfree sampling

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

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

17.1.1.4 Foci analysis

The Analysis of Foci Structure and Dynamics (AFSD), introduced by (Nelson 1996) and further expanded by (Laranjeira et al. 1998), was used in several studies on citrus diseases in Brazil. In this analysis, the data come from incidence maps where both the diseased and no-diseased trees are mapped in the 2D plane (Jesus Junior and Bassanezi 2004; Laranjeira et al. 2004).

Here is an example of an incidence map with four foci (adapted from (Laranjeira et al. 1998)). The data is organized in the wide format where the first column x is the index for the row and each column is the position of the plant within the row. The 0 and 1 represent the non-diseased and diseased plant, respectively.

```
foci <- tibble::tribble(
          ~x, ~`1`, ~`2`, ~`3`, ~`4`, ~`5`, ~`6`, ~`7`, ~`8`, ~`9`,
          1,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
 2,   1,   1,   1,   0,   0,   0,   0,   1,   0,
 3,   1,   1,   1,   0,   0,   0,   1,   1,   1,
 4,   0,   1,   1,   0,   0,   0,   0,   1,   0,
 5,   0,   1,   1,   0,   0,   0,   0,   0,   0,
 6,   0,   0,   0,   1,   0,   0,   0,   0,   0,
 7,   0,   0,   0,   0,   0,   0,   0,   0,   0,
 8,   0,   0,   0,   0,   0,   0,   0,   0,   0,
 9,   0,   0,   0,   0,   0,   1,   0,   1,   0,
10,   0,   0,   0,   0,   0,   0,   1,   0,   0,
11,   0,   1,   0,   0,   0,   1,   0,   1,   0,
12,   0,   0,   0,   0,   0,   0,   0,   0,   0
)
```

Since the data frame is in the wide format, we need to reshape it to the long format using `pivot_longer` function of the *tidyr* package before plotting using *ggplot2* package.

```
library(tidyr)

foci2 <- foci |>
  pivot_longer(2:10, names_to = "y", values_to = "i")
foci2
```

```
# A tibble: 108 x 3
       x y         i
   <dbl> <chr> <dbl>
 1     1 1         0
 2     1 2         0
 3     1 3         0
 4     1 4         0
 5     1 5         0
 6     1 6         0
 7     1 7         0
 8     1 8         0
 9     1 9         0
10     2 1         1
# i 98 more rows
```

Now we can make the plot.

```
library(ggplot2)
foci2 |>
```

```
ggplot(aes(x, y, fill = factor(i)))+
geom_tile(color = "black")+
scale_fill_manual(values = c("grey70", "darkred"))+
theme_void()+
coord_fixed()+
theme(legend.position = "none")
```

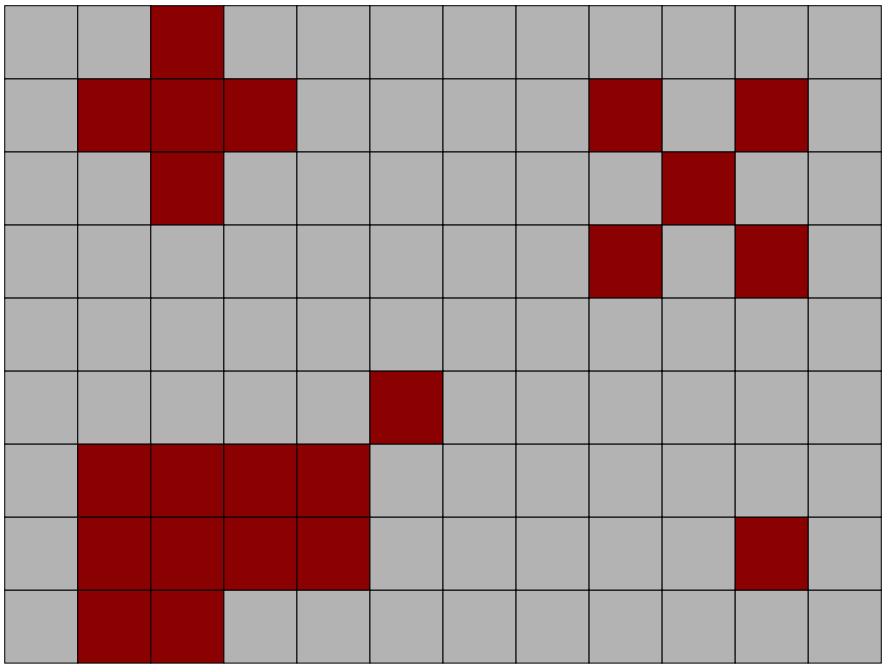


Figure 17.7: Examples of foci of plant diseases - see text for description

In the above plot, the upper left focus is composed of four diseased plants with a pattern of vertical and horizontal proximity to the central unit (or the Rook's case). The upper right focus, also with four diseased plants denotes a pattern of longitudinal proximity to the central unit (or the Bishop's case). The lower left focus is composed of 11 diseased plants with 4 rows and 6 columns occupied by the focus; the shape index of the focus (SIF) is 1.25 and the compactness index of the focus (CIF) is 0.55. The lower right is a single-unit focus.

In this analysis, several statistics can be summarized, both at the single focus and averaging across all foci in the area, including:

- Number of foci (NF) and number of single focus (NSF)
- To compare maps with different number of plants, NF and NSF can be normalized to 1000 plants as NF1000 and NSF1000

- Number of plants in each focus i (NPFi)
- Maximum number of rows of the focus i (rfi) and maximum number of columns of the focus i (cfi)
- Mean shape index of foci (meanSIF = [(fri / cfi)]/NF), where SIF values equal to 1.0 indicate isodiametrical foci; values greater than 1.0 indicate foci with greater length in the direction between the planting rows and values less than 1 indicate foci with greater length in the direction of the planting row.
- Mean compactness index of foci (meanCIF = [(NPFi/rfi*cfi)]/NF), where CIF values close to 1.0 indicate a more compact foci, that is, greater aggregation and proximity among all the plants belonging to the focus

We can obtain the above-mentioned foci statistics using the `AFSD` function of the *r4pde* package. Let's calculate for the foci2 dataset already loaded, but first we need to check whether all variables are numeric or integer.

```
str(foci2) # y was not numeric
```

```
tibble [108 x 3] (S3: tbl_df/tbl/data.frame)
 $ x: num [1:108] 1 1 1 1 1 1 1 1 2 ...
 $ y: chr [1:108] "1" "2" "3" "4" ...
 $ i: num [1:108] 0 0 0 0 0 0 0 0 1 ...
```

```
foci2$y <- as.integer(foci2$y) # transform to numeric

library(r4pde)
result_foci <- AFSD(foci2)
```

The AFSD function returns a list of three data frames. The first is a summary statistics of this analysis, together with the disease incidence (DIS_INC), for the data frame in analysis.

```
knitr::kable(result_foci[[1]])
```

stats	value
NF	4.0000000
NF1000	37.0370370
NSF	1.0000000
NSF1000	9.2592593
DIS_INC	0.2037037
mean_SIF	1.0625000

stats	value
mean_CIF	0.6652778

The second object in the list is a data frame with statistics at the focus level, including the number of rows and columns occupied by each focus as well as the two indices for each focus: shape and compactness.

```
knitr::kable(result_foci[[2]])
```

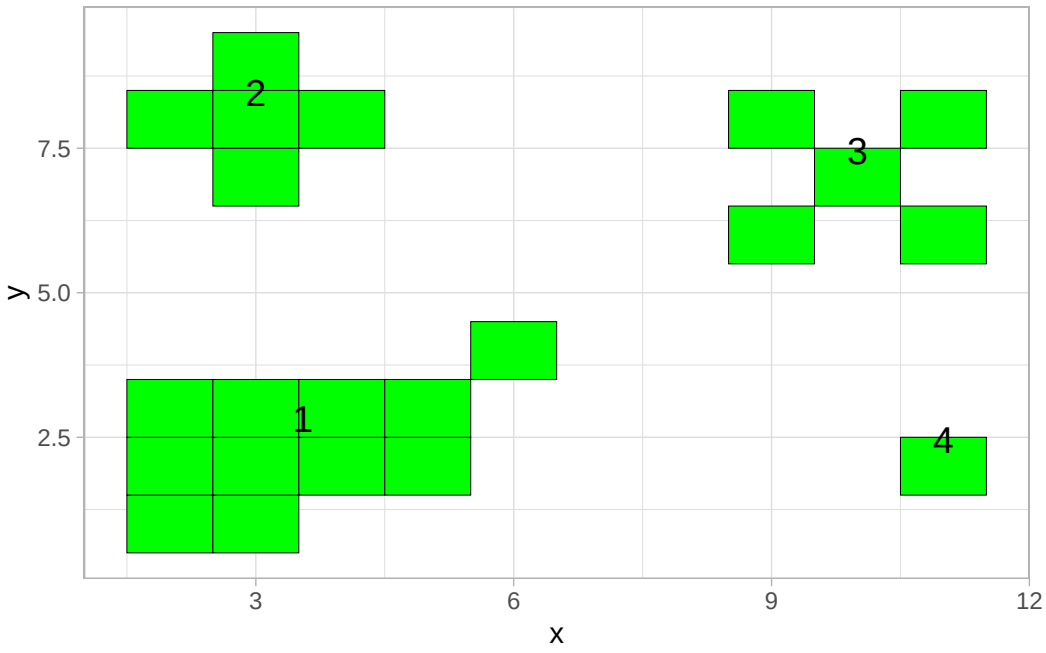
focus_id	size	rows	cols	SIF	CIF
1	11	4	5	0.8	0.5500000
2	5	3	3	1.0	0.5555556
3	5	3	3	1.0	0.5555556
4	1	1	1	1.0	1.0000000

The third object is the original data frame amended with the id for each focus which can be plotted and labelled (the focus ID) using the `plot_AFSD()` function.

```
foci_data <- result_foci[[3]]
DT::datatable(foci_data)
```

The plot shows the ID for each focus.

```
plot_AFSD(foci_data)+
  theme_light()
```

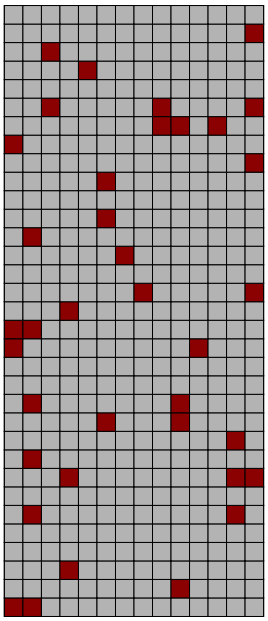


We will now analyse an actual data set from the *epiphy* package. The data describe the incidence of tomato spotted wilt virus (TSWV) disease in field trials. There are two years in the data set. We will work with the data from 1928 when six assessments were made in time. We will first work with time 1 by using `filter()` function of the *dplyr* package.

```
library(epiphy)
tswv_1928 <- tomato_tswv$field_1928
df1 <- tswv_1928 |>
  filter(t == 1) |> # filter time 1
  select(x, y, i) # select only three variables
```

Follows the incidence map of the area at time 1.

```
df1 |>
  ggplot(aes(x, y, fill = factor(i)))+
  geom_tile(color = "black")+
  theme_void()+
  scale_fill_manual(values = c("grey70", "darkred"))+
  coord_fixed()+
  theme(legend.position = "none")
```



Now we can run the AFSD function and obtain the statistics.

```
result_df1 <- AFSD(df1)

knitr::kable(result_df1[[1]])
```

stats	value
NF	33.0000000
NF1000	17.8571429
NSF	8.0000000
NSF1000	4.3290043
DIS_INC	0.0784632
mean_SIF	1.1482323
mean_CIF	0.7977633

This analysis is usually applied to multiple maps and the statistics are visually related to the incidence in the area in a scatter plot. Let's calculate the statistics for all five times of the data frame where we will keep now the time variable in the dataframe and split it by time before applying the function. We can do it using the `map` function of the *purrr* package.

```
library(purrr)

df_all <- tomato_tswv$field_1928

# Split the dataframe by 'time'
df_split <- split(df_all, df_all$t)

# Apply the AFSD function to each split dataframe
results <- map(df_split, AFSD)
```

We can check the summary results for time 2 and time 3.

```
time2 <- data.frame(results[[2]][1])
knitr::kable(time2)
```

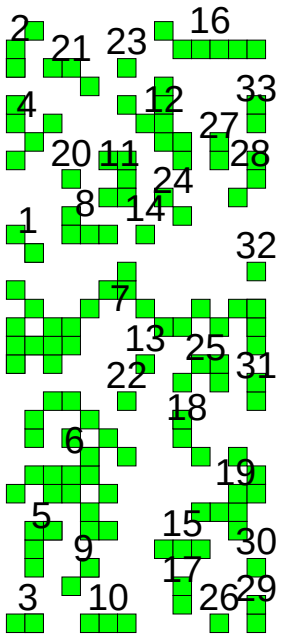
stats	value
NF	2.0000000
NF1000	1.0822511
NSF	0.0000000
NSF1000	0.0000000
DIS_INC	0.2364719
mean_SIF	1.7121212
mean_CIF	1.1352814

```
time3 <- data.frame(results[[3]][1])
knitr::kable(time3)
```

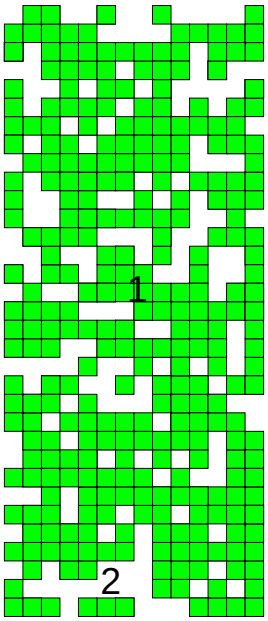
stats	value
NF	1.0000000
NF1000	0.5411255
NSF	0.0000000
NSF1000	0.0000000
DIS_INC	0.4085498
mean_SIF	0.4242424
mean_CIF	1.6341991

```
# Plot the results to see the two foci in time 1
plot_AFSD(results[[1]][[3]])+
```

```
  theme_void()+
  coord_fixed()
```




```
# Plot time 2
plot_AFSD(results[[2]][[3]])+
  theme_void()+
  coord_fixed()
```



17.1.2 Point pattern analysis

Point pattern analysis involves the study of the spatial arrangement of points in a two-dimensional space. In its simplest form, one can visualize this as a scatter plot on a map, where each point represents an event, object, or entity in space. For example, the points might represent the locations of diseased plants in a population.

The easiest way to visualize a 2-D point pattern is to produce a map of the locations, which is simply a scatter plot but with the provision that the axes are equally scaled. However, while the visualization can provide a basic understanding of the spatial distribution, the real power of point pattern analysis lies in the quantitative methods that allow one to analyze the distribution in a more detailed and systematic way. These methods help to identify whether the points are randomly distributed, clustered (points are closer together than expected by chance), or regularly spaced (points are more evenly spaced than expected by chance). This analysis can provide insights into underlying processes that might explain the observed patterns.

Let's work with two simulated datasets that were originally generated to produced a random or an aggregated (clustered) pattern.

```
library(r4pde)
rand <- SpatialRandom
aggr <- SpatialAggregated
```

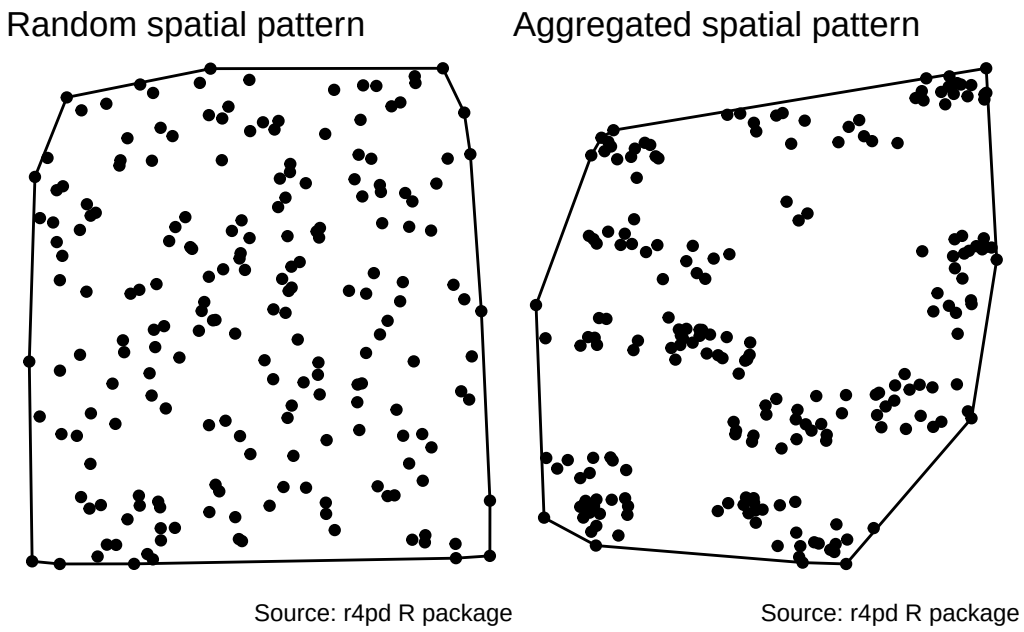
In order to create a polygon with the most extreme points, we can use the `chull()` function to find the convex hull, which will give us the indices of the points that form the smallest convex polygon that contains all the points in our data set.

```
hull_indices_rand <- chull(rand)
# Add these indices as a new column to the data frame
rand$hull <- FALSE
rand$hull[hull_indices_rand] <- TRUE

hull_indices_aggr <- chull(aggr)
# Add these indices as a new column to the data frame
aggr$hull <- FALSE
aggr$hull[hull_indices_aggr] <- TRUE
```

The two dataframes has two variables each. Let's produce 2-D map.

```
prand <- rand |>
  ggplot(aes(x, y))+
  geom_polygon(data = rand[hull_indices_rand, ], aes(x, y), fill = NA, color = 'black') +
  geom_point()+
  scale_color_manual(values = c("black", NA))+
  coord_fixed()+
  coord_flip()+
  theme_void()+
  theme(legend.position = "none")+
  labs (title = "Random spatial pattern",
        x = "Latitude",
        y = "Longitude",
        caption = "Source: r4pd R package")

paggr <- aggr |>
  ggplot(aes(x, y))+
  geom_polygon(data = aggr[hull_indices_aggr, ], aes(x, y), fill = NA, color = 'black') +
  geom_point()+
  scale_color_manual(values = c("black", NA))+
  coord_fixed()+
  coord_flip()+
  theme_void()+
  theme(legend.position = "none")+
  labs (title = "Aggregated spatial pattern",
        x = "Latitude",
        y = "Longitude",
```

```
        caption = "Source: r4pd R package")

library(patchwork)
prand | paggr
```



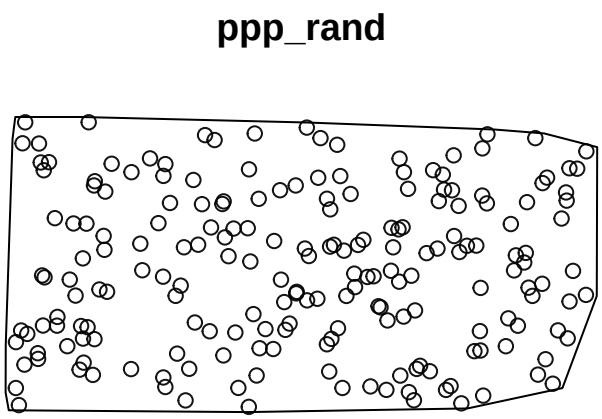
17.1.2.1 Quadrat count

In the Quadrat Count method, the study region is divided into a regular grid of smaller, equally-sized rectangular or square subregions known as quadrats. For each quadrat, the number of points falling inside it is counted. If the points are uniformly and independently distributed across the region (i.e., random), then the number of points in each quadrat should follow a Poisson distribution. If the variance of the counts is roughly equal to the mean of the counts, then the pattern is considered to be random. If the variance is greater than the mean, it suggests that the pattern is aggregated or clumped.

Using the {spatstats} package, we first need to create a ppp object which represents a point pattern. This is the primary object type in spatstat for point patterns.

```
library(spatstat)
### Create window
window_rand <- ripras(rand$x, rand$y)
```

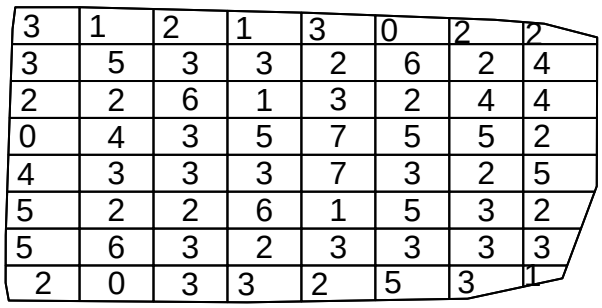
```r
# create the point pattern object
ppp_rand <- ppp(rand$x, rand$y, window_rand)
plot(ppp_rand)
```



Using the `quadratcount` function, we can divide the study region into a grid and count the number of points in each cell:

```r
## Quadrat count 8 x 8
qq <- quadratcount(ppp_rand,8,8, keepempty = TRUE)

# plot the quadrat count
plot(qq)
```

qq



To determine whether the observed distribution of points is consistent with a random Poisson process, we can use the`quadrat.test` function:

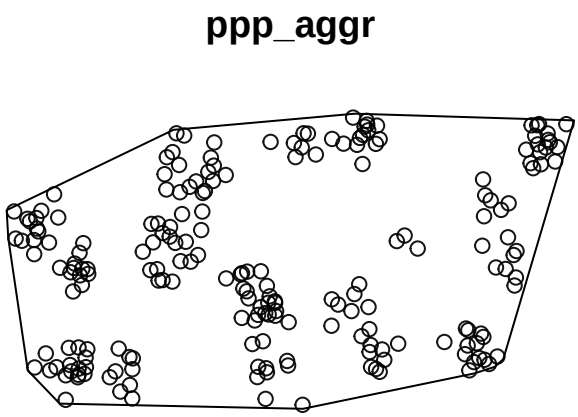
```
# Quadrat test
qt <- quadrat.test(qq, alternative="clustered", method="M")
qt
```

```
    Conditional Monte Carlo test of CSR using quadrat counts
    Test statistic: Pearson X2 statistic

data:
X2 = 48.441, p-value = 0.92
alternative hypothesis: clustered

Quadrats: 64 tiles (irregular windows)
```

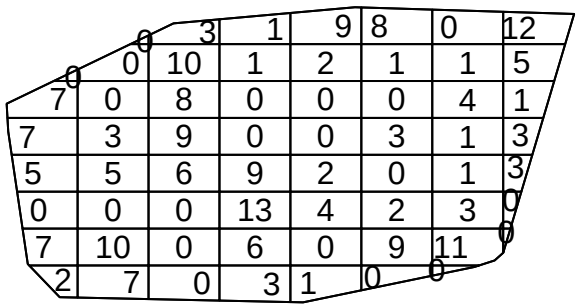
The result will give a Chi-squared statistic and a p-value. If the p-value is very low, then the pattern is likely not random. Keep in mind that the choice of the number and size of quadrats can affect the results. It's often helpful to try a few different configurations to ensure robust conclusions.

Lets repeat this procedure for the situation of aggregated data.

```
### Create window
window_aggr <- ripras(aggr$x, aggr$y)

# create the point pattern object
ppp_aggr <- ppp(aggr$x, aggr$y, window_aggr)
plot(ppp_aggr)
```




```
## Quadrat count 8 x 8
qq_aggr <- quadratcount(ppp_aggr,8,8, keepempty=TRUE)

# plot the quadrat count
plot(qq_aggr)
```




```
# Quadrat test
qt_aggr <- quadrat.test(qq, alternative="clustered", method="M")
qt_aggr
```

```
    Conditional Monte Carlo test of CSR using quadrat counts
    Test statistic: Pearson X2 statistic

data:
X2 = 48.441, p-value = 0.9115
alternative hypothesis: clustered

Quadrats: 64 tiles (irregular windows)
```

17.1.2.2 Spatial KS test

The Spatial Kolmogorov-Smirnov (KS) Test is a method to assess the goodness-of-fit of a given point pattern to the assumptions of Complete Spatial Randomness (CSR) Baddeley et al. (2005). Essentially, this means that it helps in determining whether a set of spatial points is distributed randomly or if there is some underlying pattern or interaction.

However, unlike other goodness-of-fit tests in the spatial context, the Spatial KS Test leverages the values of a spatial covariate at the observed data points and contrasts them against the expected distribution of the same covariate under the assumption of CSR. The idea behind this test is to check if there's any difference between the observed distribution of a spatial covariate's values and the expected distribution if the points were distributed in a completely spatial random fashion.

Key points of the test are:

- **Covariate**: For this test, a spatial covariate must be chosen. This is a spatially varying feature or value that might influence the point process. Examples include elevation, soil quality, or distance from a specific feature. The distribution of this covariate's values at the observed data points is the crux of the test.
- **Comparison with CSR**: The observed distribution of the spatial covariate's values at the data points is juxtaposed with what would be expected under the CSR model. The CSR model posits that points are distributed purely by chance, without any underlying structure or influence.
- **Methodology**: The test employs a classical goodness-of-fit approach to contrast the observed and expected distributions. Specifically, it utilizes the Kolmogorov-Smirnov statistic, a non-parametric measure to gauge the similarity between two distributions.

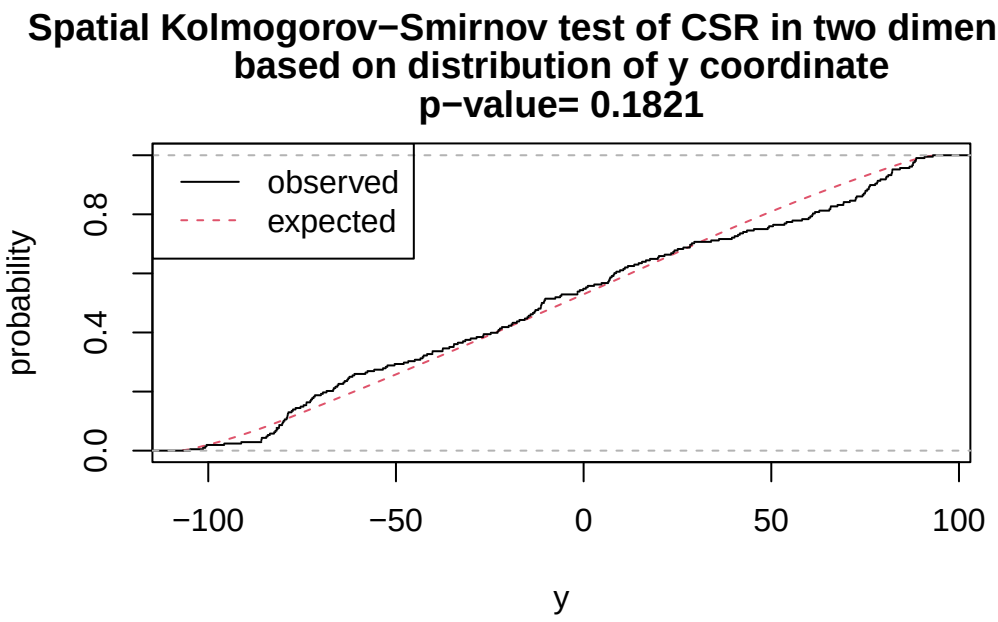
As example, we may want to select spatial coordinates themselves (like `x`, `y`, or both) as the covariates. This means the test will assess how the observed distribution of x-coordinates (or y-coordinates or both) of the data points compares to what would be anticipated under CSR.

Let's test for the aggregated data.

```
# y as covariate
ks_y <- cdf.test(ppp_aggr, test="ks", "y", jitter=FALSE)
ks_y
```

```
    Spatial Kolmogorov-Smirnov test of CSR in two dimensions

data:  covariate 'y' evaluated at points of 'ppp_aggr'
     and transformed to uniform distribution under CSR
D = 0.075883, p-value = 0.1821
alternative hypothesis: two-sided
```

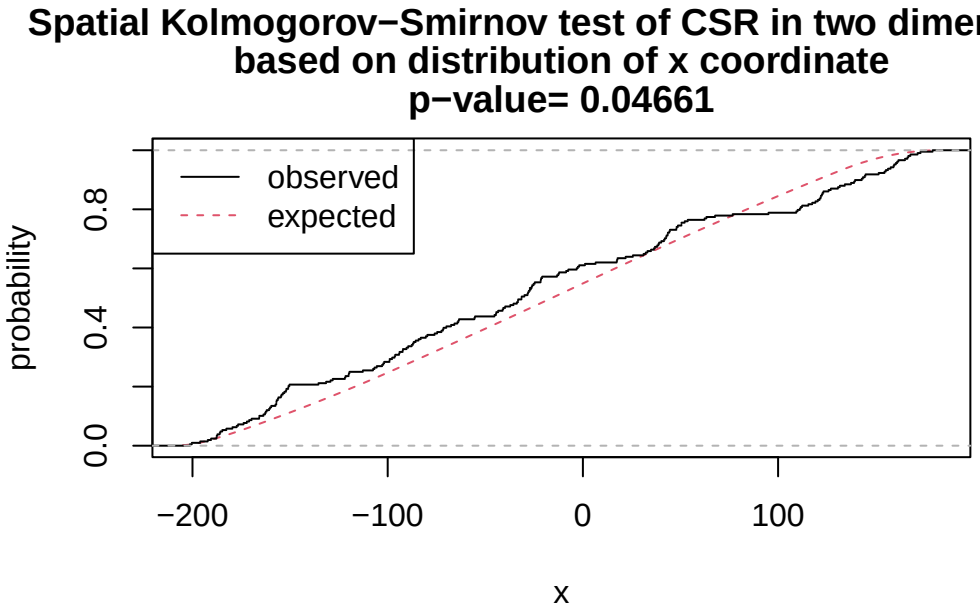
```
plot(ks_y)
```




```
# x as covariate
ks_x <- cdf.test(ppp_aggr, test="ks", "x", jitter=FALSE)
ks_x
```

```
    Spatial Kolmogorov-Smirnov test of CSR in two dimensions

data:  covariate 'x' evaluated at points of 'ppp_aggr'
     and transformed to uniform distribution under CSR
D = 0.09506, p-value = 0.04661
alternative hypothesis: two-sided
```

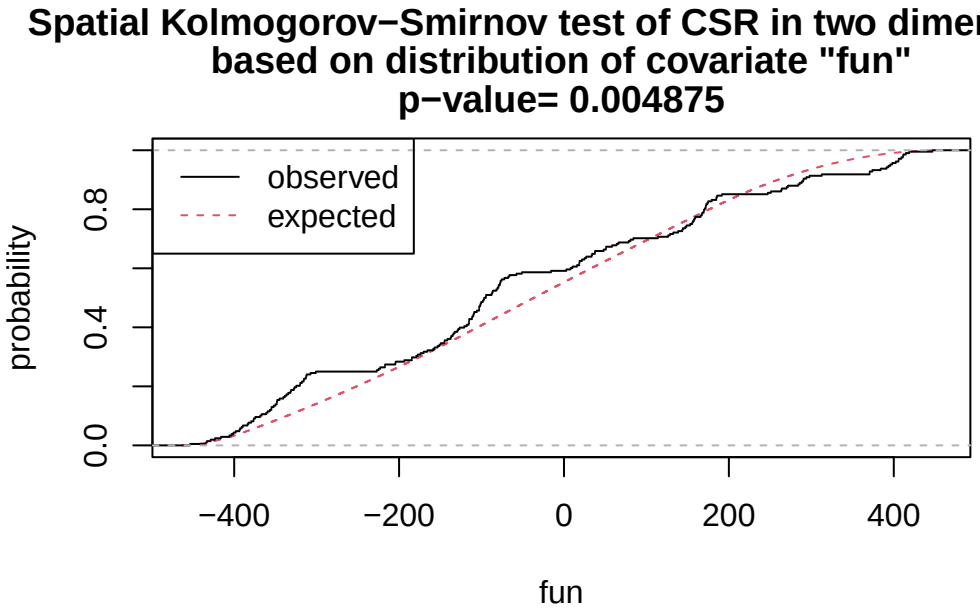
```
plot(ks_x)
```




```
# x and y as covariates
fun <- function(x,y){2* x + y}
ks_xy <- cdf.test(ppp_aggr, test="ks", fun, jitter=FALSE)
ks_xy
```

```
    Spatial Kolmogorov-Smirnov test of CSR in two dimensions

data:  covariate 'fun' evaluated at points of 'ppp_aggr'
     and transformed to uniform distribution under CSR
D = 0.12026, p-value = 0.004875
alternative hypothesis: two-sided
```

```
plot(ks_xy)
```



As shown above, we have sufficient evidence to reject the null hypothesis of complete spatial randomness.

17.1.2.3 Distance based

17.1.2.3.1 Ripley's K

A spatial point process is a set of irregularly distributed locations within a defined region which are assumed to have been generated by some form of stochastic mechanism. The **K function**, a.k.a. Ripley's K-function, is a statistical measure used in spatial analysis to examine the spatial distribution of a single type of point in a given area. Named after its developer, the British statistician B.D. Ripley, the K-function measures the expected number of points within a given distance of an arbitrary point, assuming homogeneous intensity (a constant probability of a point occurring in a particular place).

To describe it simply: imagine you have a map of diseased trees in a forest, and you select a tree at random. The K-function helps you answer the question: "How many other diseased trees do I expect to find within a certain distance from the diseased tree I've chosen?"

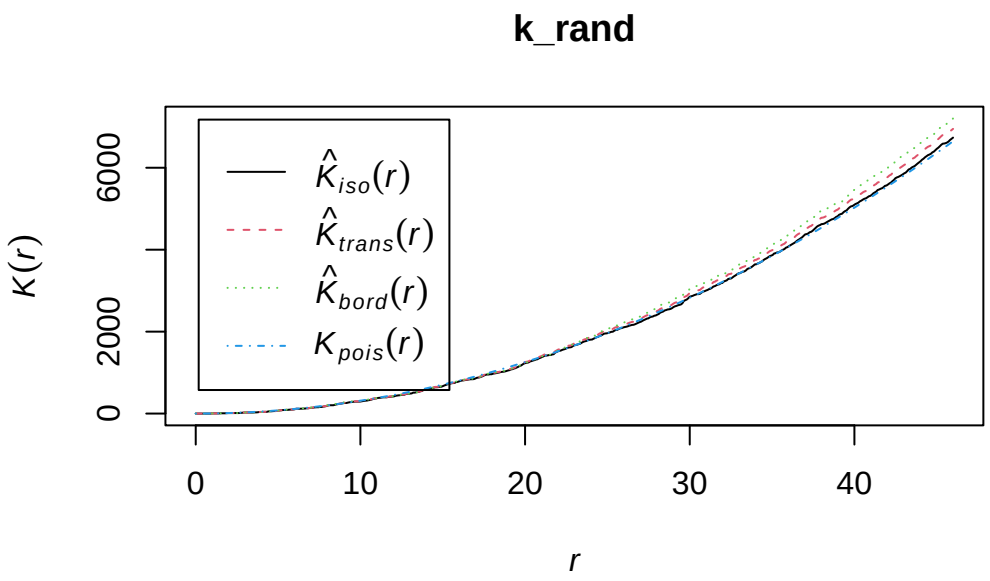
The K-function is often used to identify and analyze patterns within spatial data, such as clustering, randomness, or regularity (dispersion). It is particularly useful because it looks at the distribution at all scales (distances) simultaneously. To interpret the results of Ripley's K-function:

1. **Random distribution**: If the points (like trees in our example) are randomly distributed, the plot of the K-function will be a straight line at a 45-degree angle.
2. **Clustered distribution**: If the points are clustered (grouped closer together than you'd expect by chance), the plot will be above the 45-degree line of the random expectation.
3. **Regular or dispersed distribution**: If the points are regularly spaced or dispersed (further apart than you'd expect by chance), the plot will be below the 45-degree line.

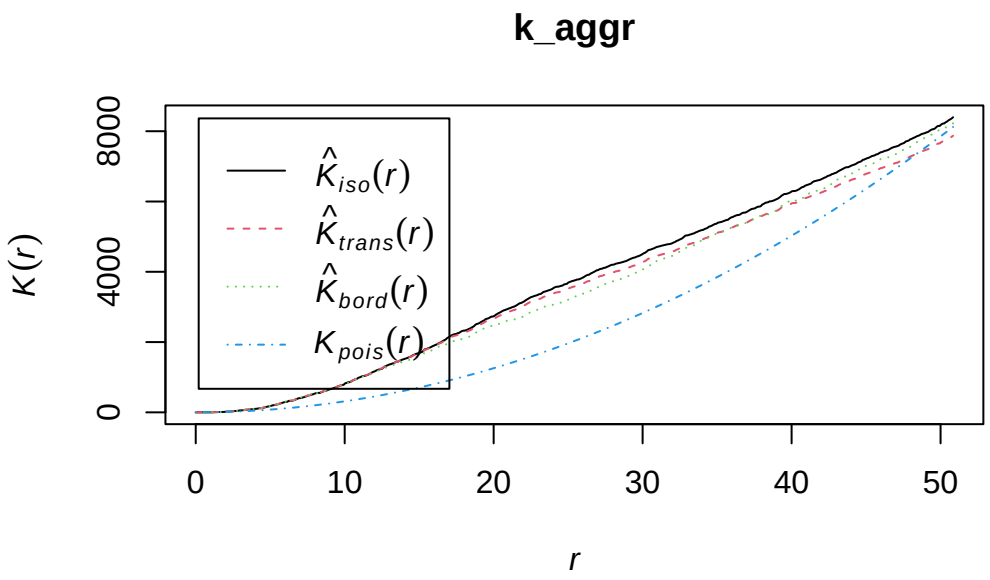
Ripley's K checks the density of diseased units in each area by the variance as a function of radial distances (r) from the diseased unit, hence *K(r)*. If the spatial localization of a diseased unit is independent, the process is random in space.

Let's use the `Kest` function of the spatstat package to obtain *K(r)*.

```
k_rand <- Kest(ppp_rand)
plot(k_rand)
```




```
k_aggr <- Kest(ppp_aggr)
plot(k_aggr)
```



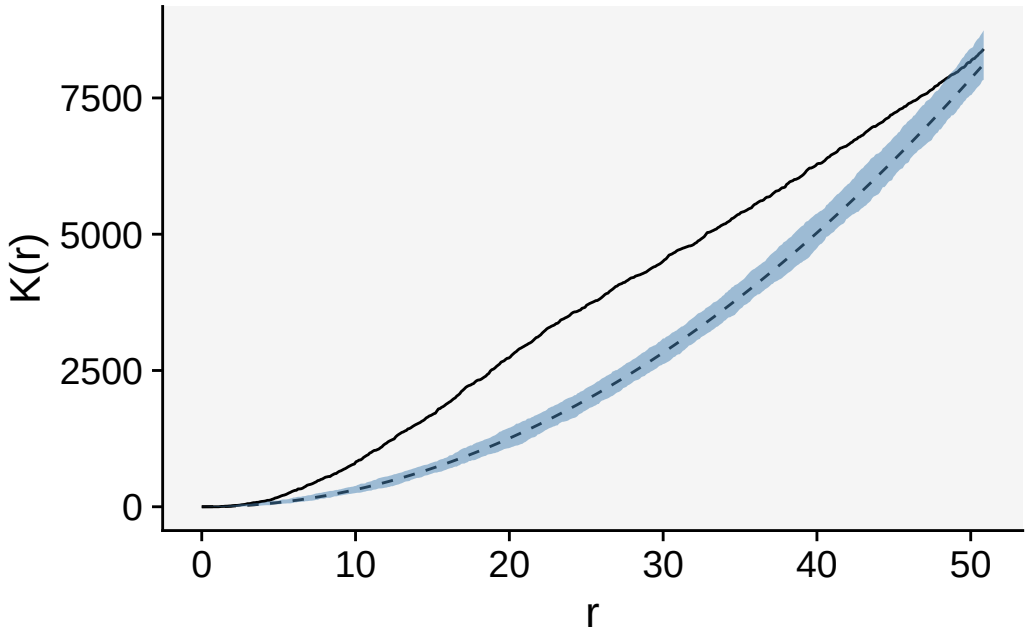
The `envelope` function performs simulations and computes envelopes of a summary statistic based on the simulations. The envelope can be used to assess the goodness-of-fit of a point process model to point pattern data (Baddeley et al. 2014). Let's simulate the envelope and plot the values using ggplot. Because observed *K(r)* (solid line) lied outside the simulation envelope, aggregation was detected.

```
ke <- envelope(ppp_aggr, fun = Kest)
```

```
Generating 99 simulations of CSR  ...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 2
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98,  99.

Done.
```

```
data.frame(ke) |>
  ggplot(aes(r, theo))+
  geom_line(linetype =2)+
  geom_line(aes(r, obs))+
  geom_ribbon(aes(ymin = lo, ymax = hi),
              fill = "steelblue", alpha = 0.5)+
```

```
  labs(y = "K(r)", x = "r")+
  theme_r4pde(font_size = 16)
```



`mad.test` performs the 'global' or 'Maximum Absolute Deviation' test described by Ripley (1977, 1981). See (Baddeley et al. 2014). This performs hypothesis tests for goodness-of-fit of a point pattern data set to a point process model, based on Monte Carlo simulation from the model.

```
# Maximum absolute deviation test
mad.test(ppp_aggr, Kest)
```

```
Generating 99 simulations of CSR  ...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 6
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98,  99.

Done.
```

```
  Maximum absolute deviation test of CSR
  Monte Carlo test based on 99 simulations
```

```
    Summary function: K(r)
    Reference function: theoretical
    Alternative: two.sided
    Interval of distance values: [0, 50.8401070056579]
    Test statistic: Maximum absolute deviation
    Deviation = observed minus theoretical

data:  ppp_aggr
mad = 1765.1, rank = 1, p-value = 0.01
```

```
mad.test(ppp_rand, Kest)
```

```
Generating 99 simulations of CSR  ...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 6
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98,  99.

Done.
```

```
    Maximum absolute deviation test of CSR
    Monte Carlo test based on 99 simulations
    Summary function: K(r)
    Reference function: theoretical
    Alternative: two.sided
    Interval of distance values: [0, 45.9916850622588]
    Test statistic: Maximum absolute deviation
    Deviation = observed minus theoretical

data:  ppp_rand
mad = 122.15, rank = 86, p-value = 0.86
```

17.1.2.3.2 O-ring statistics

Another statistics that can be used is the **O-ring** statitics which are used in spatial analysis to quantify and test the degree of interaction between two types of spatial points (Wiegand and A. Moloney 2004). The name derives from the method of placing a series of concentric circles (O-rings) around each point of type 1 and counting how many points of type 2 fall within each ring. The plot generated by O-ring statistics is called an O-ring plot or an O-function plot. It plots the radius of the rings on the x-axis and the estimated intensity of points of type 2 around points of type 1 on the y-axis.

Interpreting the plot is as follows:

1. **Random pattern**: If points of type 2 are randomly distributed around points of type 1, the O-ring plot will be a flat line. This means that the intensity of points of type 2 does not change with the distance to points of type 1.
2. **Aggregation or clustering**: If points of type 2 are aggregated around points of type 1, the O-ring plot will be an upward-sloping curve. This indicates that the intensity of points of type 2 increases with proximity to points of type 1.
3. **Dispersion**: If points of type 2 are dispersed away from points of type 1, the O-ring plot will be a downward-sloping curve. This shows that the intensity of points of type 2 decreases as you get closer to points of type 1.

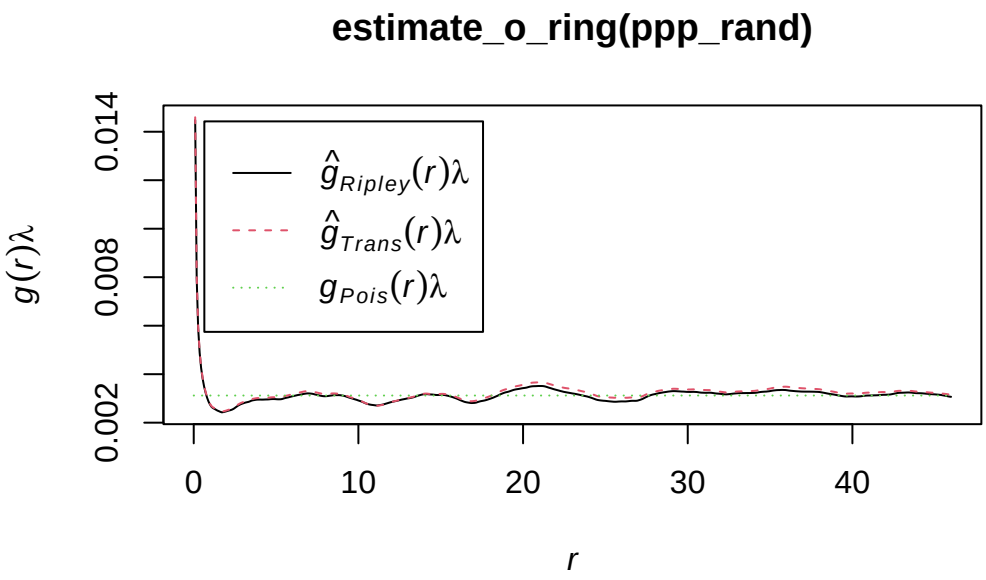
The O-ring plot often includes a confidence envelope. If the O-ring statistic falls within this envelope, it suggests that the observed pattern could be the result of random spatial processes. If it falls outside the envelope, it suggests that the pattern is not random. Therefore, to decide whether a pattern is aggregated or random using O-ring statistics:

- Look at the shape of the O-ring plot.
- Compare the O-ring statistic to the confidence envelope.

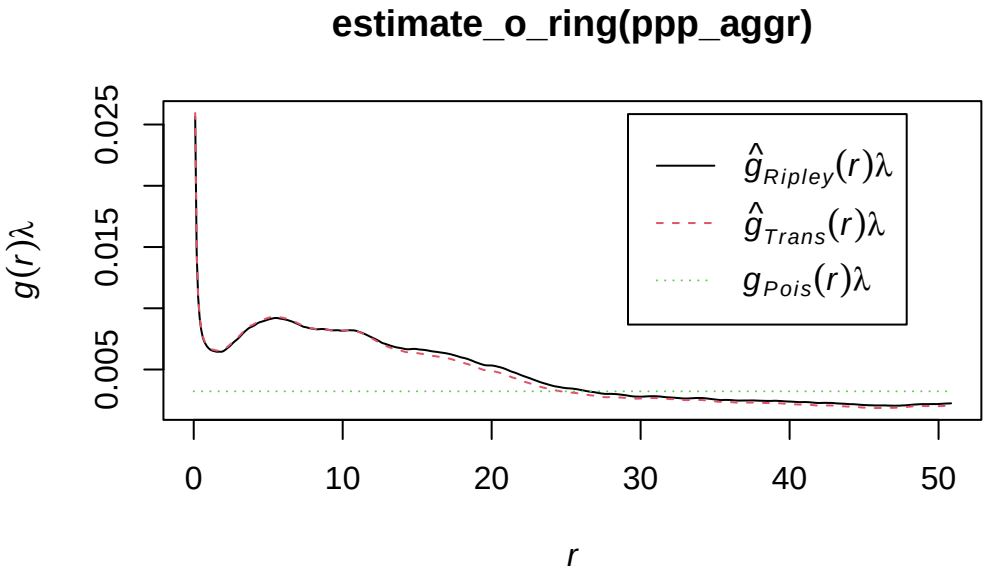
An aggregated pattern will show an increasing curve that lies outside the confidence envelope, indicating that the density of type 2 points is higher close to type 1 points. On the other hand, a random pattern will show a flat line that lies within the confidence envelope, indicating no significant difference in the density of type 2 points around type 1 points at varying distances.

In R, we can use the `estimate_o_ring()` function of the *onpoint* package. We will use the point pattern object `ppp_fw` used in the previous examples

```
library(onpoint)
plot(estimate_o_ring(ppp_rand))
```

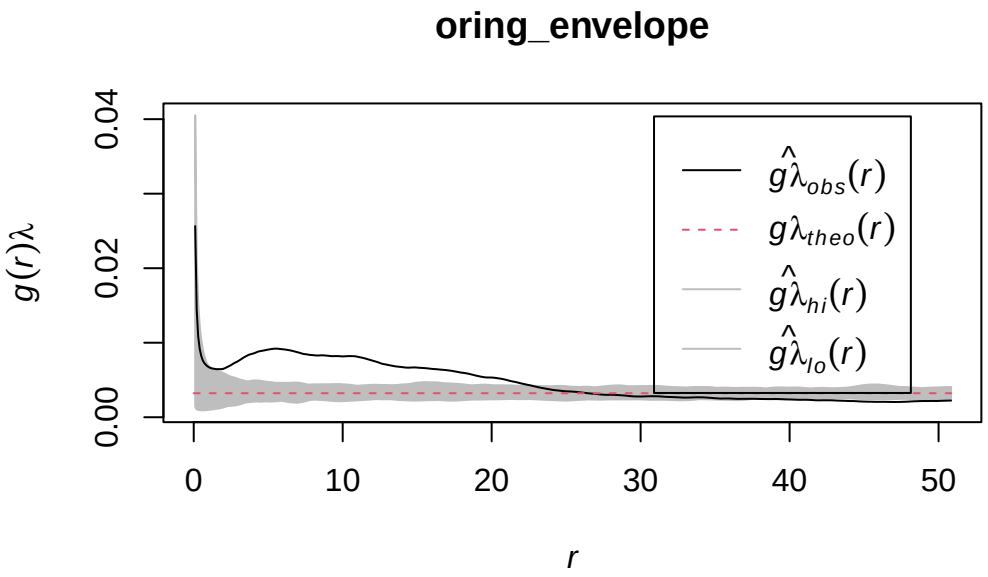



```
plot(estimate_o_ring(ppp_aggr))
```



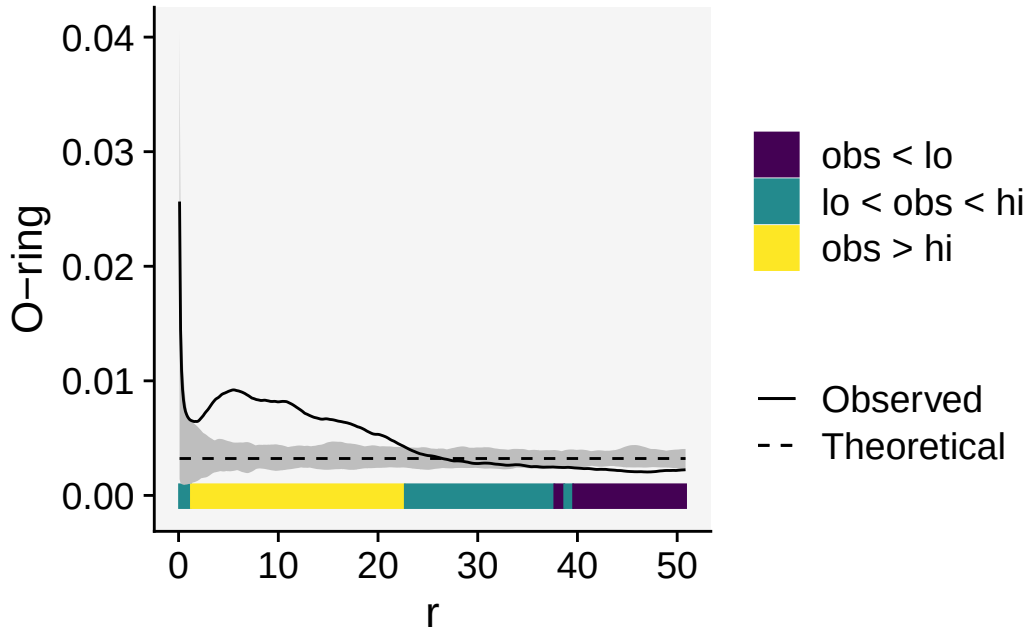
The function can be used in combination with `spatstat`'s `envelope()` function.

```
oring_envelope <- envelope(ppp_aggr, fun = estimate_o_ring, nsim = 199, verbose = FALSE)
plot(oring_envelope)
```



To plot simulation envelopes using quantum plots (Esser et al. 2014), just pass an `envelope` object as input to `plot_quantums()`.

```
plot_quantums(oring_envelope, ylab = "O-ring")+
  theme_r4pde()
```



17.1.3 Grouped data

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

- Autocorrelation (known as Moran's I)
- Semivariance
- SADIE (an alternative approach to autocorrelation.)

17.1.3.1 Autocorrelation

Spatial autocorrelation analysis provides a quantitative assessment of whether a large value of disease intensity in a sampling unit makes it more (positive autocorrelation) or less (negative auto- correlation) likely that neighboring sampling units tend to have a large value of disease intensity (Madden et al. 2007b).

We will illustrate the basic concept by reproducing the example provided in page 264 of the chapter on spatial analysis (Madden et al. 2007b), which was extracted from table 11.3 of Campbell and Madden. L. (1990). The data represent a single transect with the number of *Macrophomia phaseolina* propagules per 10 g air-dry soil recorded in 16 contiguous quadrats across a field.

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

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

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

```
mp |>
  ggplot(aes(i, y)) +
  geom_col(fill = "darkred") +
  theme_r4pde()+
  labs(
    x = "Relative position within a transect",
    y = "Number of propagules",
    caption = "Source: Campbell and Madden (1990)"
  )
```

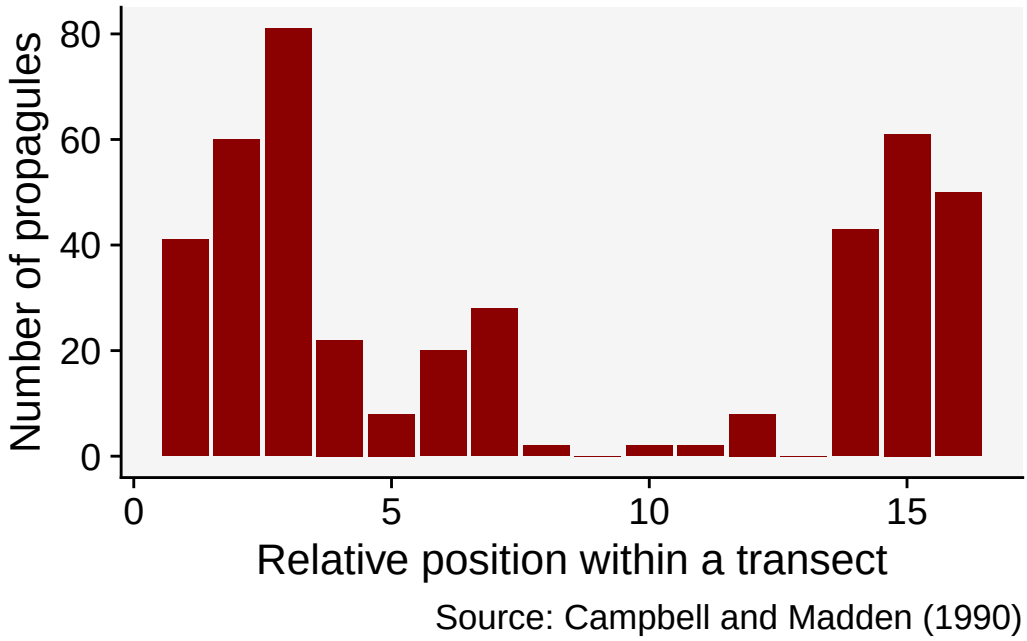


Figure 17.8: Number of propagules of Macrophomina phaseolina in the soil at various positions within a transect

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

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

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

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

Let's store the results in a data frame to facilitate visualization.

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

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

And now the plot known as autocorrelogram.

```
ac_mp_dat |>
  ggplot(aes(index, ac_mp.acf, label = round(ac_mp.acf, 3))) +
  geom_col(fill = "darkred") +
  theme_r4pde()+
  geom_text(vjust = 0, nudge_y = 0.05) +
  scale_x_continuous(n.breaks = 6) +
  geom_hline(yintercept = 0) +
  labs(x = "Distance lag", y = "Autocorrelation coefficient")
```

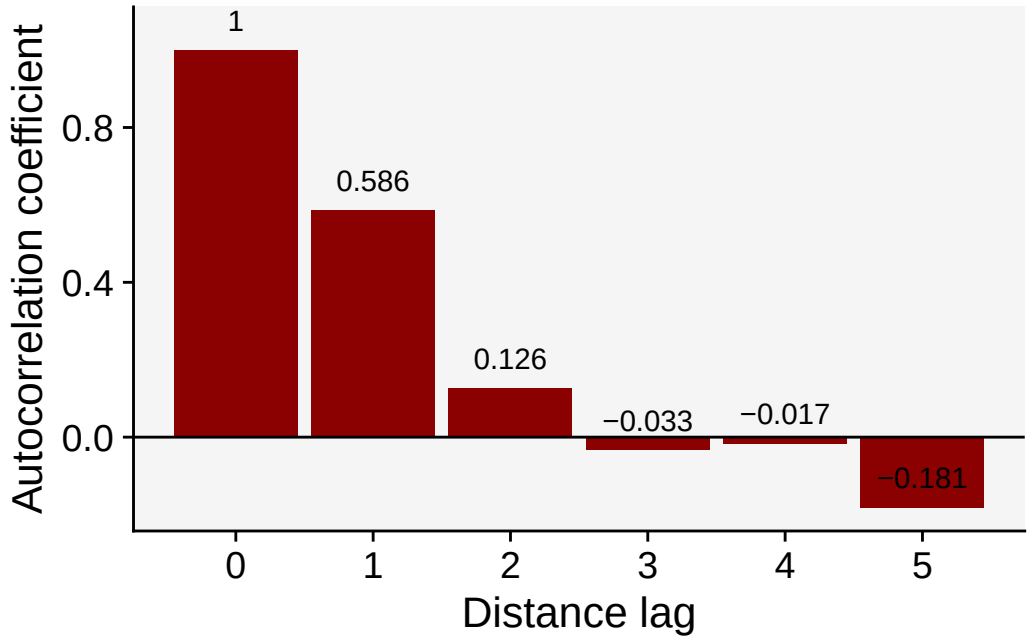


Figure 17.9: Autocorrelogram for the spatial distribution of Macrophomina phaseolina in soil

The values we obtained here are not the same but quite close to the values reported in Madden

et al. (2007d). For the transect data, the calculated coefficients in the book example for lags 1, 2 and 3 are 0.625, 0.144, and - 0.041. The conclusion is the same, the smaller the distance between sampling units, the stronger is the correlation between the count values.

17.1.3.1.1 Moran's I

The method above is usually referred to Moran's I (Moran 1950), a widely-used statistic to measure spatial autocorrelation in spatial datasets. The presence of spatial autocorrelation implies that nearby locations exhibit more similar (positive autocorrelation) or dissimilar (negative autocorrelation) values than would be expected under random arrangements.

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

```
epi <- read_csv("https://raw.githubusercontent.com/emdelponte/epidemiology-R/main/data/xu-
epi1 <- epi |>
  pivot_longer(2:13,
               names_to = "y",
               values_to = "n") |>
  pull(n)
```

The {spdep} package in R provides a suite of functions for handling spatial dependence in data, and one of its main functions for assessing spatial autocorrelation is **`moran()`**. The **`moran()`** function calculates Moran's I statistic for a given dataset. It requires two primary inputs: 1) A **Numeric Vector**: This vector represents the values for which spatial autocorrelation is to be measured. Typically, this could be a variable like population density, temperature, or any other spatial attribute; and 2) A **Spatial Weights Matrix**: This matrix represents the spatial relationships between the observations. It can be thought of as defining the "neighbors" for each observation. The {spdep} package provides functions to create various types of spatial weights, such as contiguity-based weights or distance-based weights. Let's load the library and start with the procedures.

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

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

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

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

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

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

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

```
$I
[1] 0.05818595

$K
[1] 2.878088
```

The `$I` is Moran's I and `$K` output is the sample kurtosis of x, or a measure of the "tailedness" of the probability distribution of a real-valued random variable.

The interpretations for Moran's I is as follows: A positive Moran's I indicates positive spatial autocorrelation. Nearby locations tend to have similar values, while a Negative Moran's I suggests negative spatial autocorrelation. Neighboring locations have dissimilar values. Moran's close to zero indicates no spatial autocorrelation. The distribution appears random.

In the context of Moran's I and spatial statistics, kurtosis of the data (**x**) can provide additional insights. For instance, if the data is leptokurtic (`$K` > 3), it might suggest that there are some hotspots or cold spots (clusters of high or low values) in the spatial dataset. On the other hand, platykurtic (`$K` < 3) data might indicate a more even spread without pronounced clusters. This information can be useful when interpreting the results of spatial autocorrelation tests and in understanding the underlying spatial structures.

17.1.3.1.2 Moran's test

The Moran's I test is a cornerstone of spatial statistics, used to detect spatial autocorrelation in data. In the realm of the **spdep** package in R, the **`moran.test()`** function is employed to perform this test. The Moran's test for spatial autocorrelation uses spatial weights matrix in weights list form. The key inputs are:

- **x**: This is a numeric vector containing the values we wish to test for spatial autocorrelation. It could be anything like population densities, number of diseased unites or severity, in the context of plant disease.
- **listw**: This represents the spatial weights, and it's in list form. The spatial weights matrix is essential for the Moran's test because it defines the "relationship" between different observations. How we define these relationships can vary: it might be based on distance (e.g., closer observations have higher weights), contiguity (e.g., observations that share a border), or other criteria.

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

```
    Moran I test under randomisation

data:  epi1
weights: col.W

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

The function will return the Moran's I statistic value, an expected value under the null hypothesis of no spatial autocorrelation, the variance, and a p-value. The p-value can be used to determine the statistical significance of the observed Moran's I value. The interpretation is as follows:

- A significant **positive Moran's I value** indicates positive spatial autocorrelation, meaning that similar values cluster together in the spatial dataset.
- A significant **negative Moran's I value** suggests negative spatial autocorrelation, implying that dissimilar values are adjacent to one another.
- If Moran's I is not statistically significant (based on the p-value), it suggests that the spatial pattern might be random, and there's no evidence of spatial autocorrelation.

In conclusion, the `moran.test()` function offers a structured way to investigate spatial autocorrelation in datasets. Spatial weights play a crucial role in this analysis, representing the spatial relationships between observations.

As before, we can construct a correlogram using the output of the `sp.correlogram()` function. Note that the figure below is very similar to the one shown in Figure 91.5 in page 269 of the book chapter (Madden et al. 2007b). Let's store the results in a dataframe.

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

```
df_correl <- data.frame(correl_I$res) |>
  mutate(lag = c(1:10))
# Show the spatial autocorrelation for 10 distance lags
round(df_correl$X1,3)
```

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

Then, we can generate the plot using *ggplot*.

```
df_correl |>
  ggplot(aes(lag, X1)) +
  geom_col(fill = "darkred") +
  theme_r4pde()+
  scale_x_continuous(n.breaks = 10) +
  labs(x = "Distance lag", y = "Spatial autocorrelation")
```

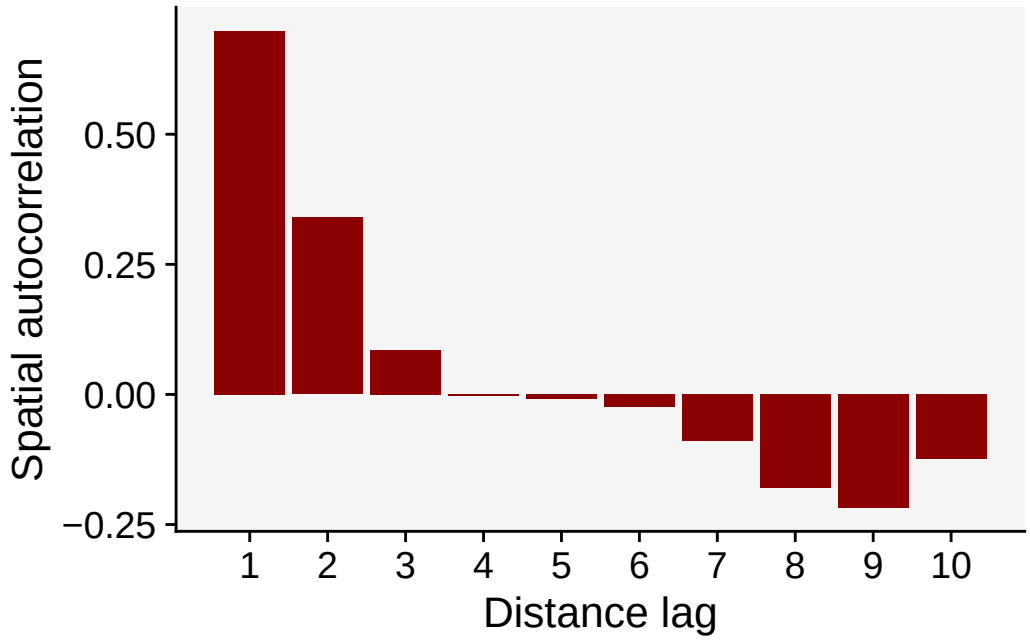


Figure 17.10: Autocorrelogram for the spatial distribution of simulated epidemics

17.1.3.2 Semivariance

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

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

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

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

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

head(epi2)
```

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

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

```
variog: computing omnidirectional variogram
```

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

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

```
# Plotting
plot(v1, xlim = c(0,15))
```

```
lines(v2, lty = 1, lwd = 2)
```

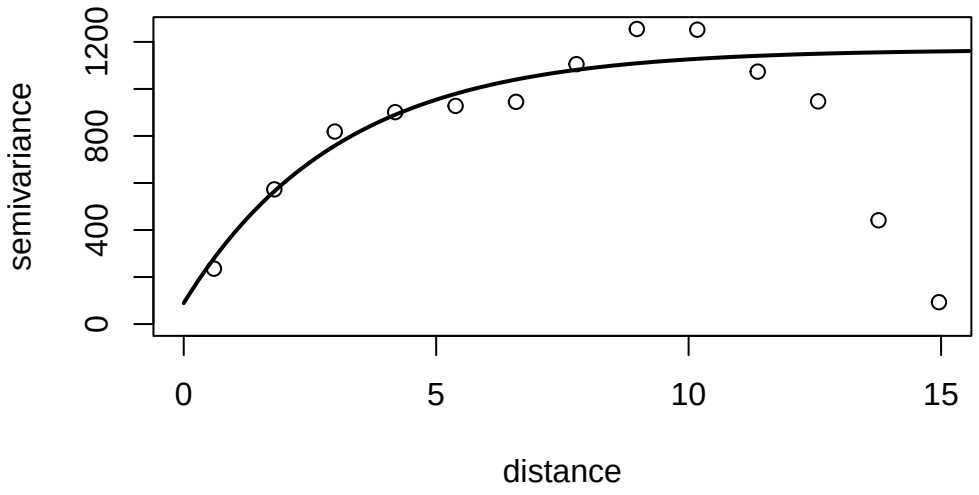


Figure 17.11: Semivariance plot for the spatial distribution simulated epidemic

17.1.3.3 SADIE

SADIE (spatial analysis by distance indices) is an alternative to autocorrelation and semivariance methods described previously, which has found use in plant pathology (Madden et al. 2007b; Xu and Madden 2004; Li et al. 2011). Similar to those methods, the spatial coordinates for the disease intensity (count of diseased individuals) or pathogen propagules values should be provided.

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

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

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

In R, we can use the `sadie()` function of the *epiphy* package (Gigot 2018). The function computes the different indices and probabilities based on the distance to regularity for the observed spatial pattern and a specified number of random permutations of this pattern. To run the analysis, the dataframe should have only three columns: the first two must be the x and y coordinates and the third one the observations. Let's continue working with the simulated epidemic dataset named `epi2`. We can map the original data as follows:

```
epi2 |>
  ggplot(aes(x, y, label = n, fill = n)) +
  geom_tile() +
  geom_text(size = 5, color = "white") +
  theme_void() +
  coord_fixed() +
  scale_fill_gradient(low = "gray70", high = "darkred")
```

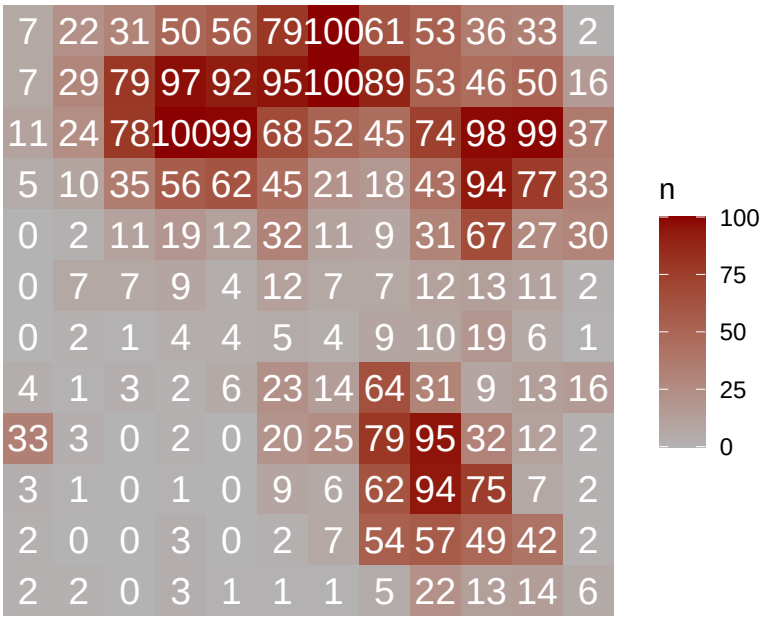


Figure 17.12: Spatial map for the number of diseased plants per quadrat (n = 144) in simulated epidemic

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

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

```
sadie_epi2
```

```
Spatial Analysis by Distance IndicEs (sadie)

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

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

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

```
summary(sadie_epi2)
```

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

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

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

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

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

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

```
plot(sadie_epi2)
```

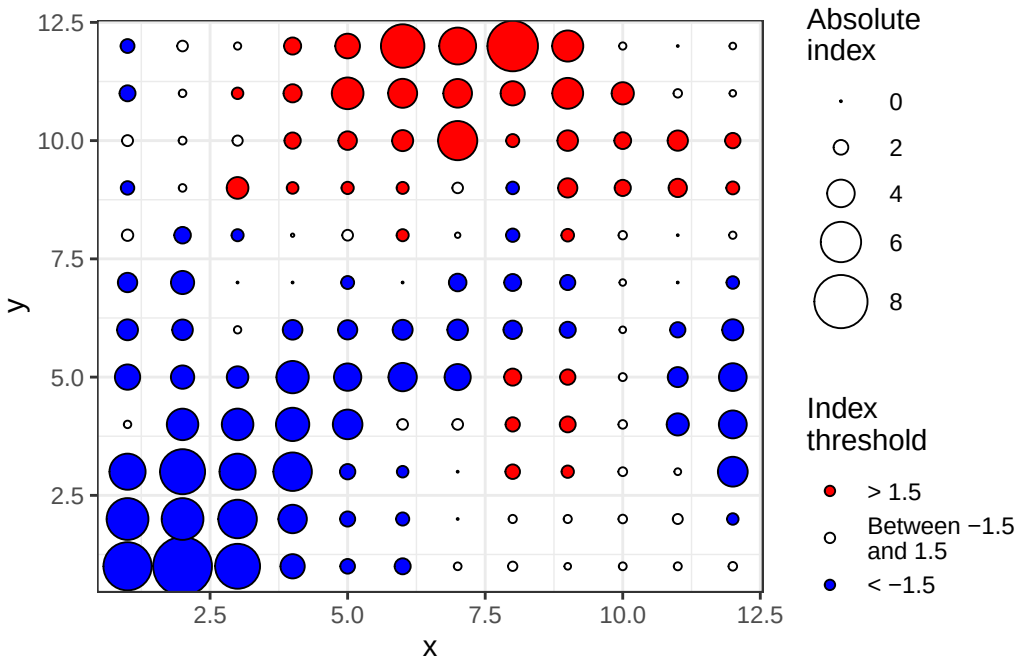


Figure 17.13: Map of the SADIE clustering indices where red identifiy patches (outflow) and blue identify gaps (inflow)

A isocline plot can be obtained by setting the `isocline` argument as `TRUE`.

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

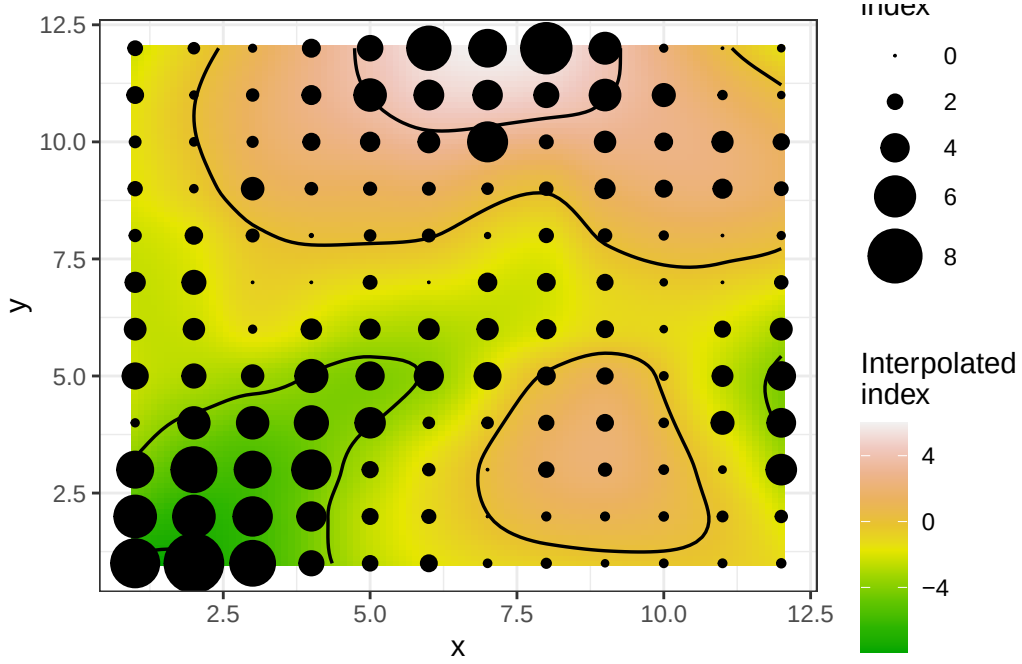


Figure 17.14: Map of the SADIE clustering indices

17.2 Sparsely sampled data

Sparsely sampled data differs significantly from intensively mapped data. While the latter provides intricate details about the spatial location of each unit, sparsely sampled data lacks this granularity. This absence means that the spatial location of individual units isn't factored into the analysis of sparsely sampled data.

In the realm of analyzing such data, particularly in understanding disease intensity, researchers often aim to characterize the range of variability in the average level of disease intensity for each sampling unit, as indicated by (Madden et al. 2007b). The methodology to analyze sparsely sampled data, especially in relation to the spatial patterns seen in plant disease epidemics, can be broadly categorized into two primary approaches:

1. **Goodness of Fit to Statistical Distributions**: This approach focuses on testing how well the observed data matches expected statistical probability distributions. By doing so, researchers can assess the likelihood that the observed data is a good fit for a particular statistical model or distribution.
2. **Indices of Aggregation Calculation**: This involves computing various metrics that can help determine the degree to which data points, or disease incidents in this case, tend to cluster or aggregate in certain areas or patterns.

Furthermore, the choice of analysis method often hinges on the nature of the data in question. For instance, data can either be in the form of raw counts or as incidences (proportions). Depending on this distinction, specific statistical distributions are presumed to best represent and describe the data. Subsequent sections will delve deeper into these methods, distinguishing between count and incidence data to provide clarity on the best analytical approach for each type.

17.2.1 Count data

17.2.1.1 Fit to distributions

Two statistical distributions can be adopted as reference for the description of random or aggregated patterns of disease data in the form of counts of infection within sampling units. Take the count of lesions on a leaf, or the count of diseased plants on a quadrat, as an example. If the presence of a lesion/diseased plant does not increase or decrease the chance that other lesions/diseased plants will occur, the *Poisson* distribution describes the distribution of lesions on the leaf. Otherwise, the *negative binomial* provides a better description.

Let's work with the previous simulation data of 144 quadrats with a variable count of diseased plants per quadrat (in a maximum of 100). Notice that we won't consider the location of each quadrat as in the previous analyses of intensively mapped data. We only need the vector with the number of infected units per sampling unit.

The *epiphy* package provides a function called `fit_two_distr()`, which allows fitting these two distribution for count data. In this case, either randomness assumption (Poisson distributions) or aggregation assumption (negative binomial) are made, and then, a goodness-of-fit comparison of both distributions is performed using a log-likelihood ratio test. The function requires a data frame created using the `count()` function where the number of infection units is designated as `i`. It won't work with a single vector of numbers. We create the data frame using:

```
data_count <- epi2 |>
  mutate(i = n) |>  # create i vector
  epiphy::count()   # create the map object of count class
```

We can now run the function that will look fo the the vector `i`. The function returns a list of four components including the outputs of the fitting process for both distribution and the result of the log-likelihood ratio test, the `llr`.

```
fit_data_count <- fit_two_distr(data_count)
summary(fit_data_count)
```

```
Fitting of two distributions by maximum likelihood
for 'count' data.
Parameter estimates:

(1) Poisson (random):
       Estimate  Std.Err Z value    Pr(>z)
lambda 27.85417  0.43981  63.333 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(2) Negative binomial (aggregated):
       Estimate    Std.Err Z value    Pr(>z)
k     0.6327452  0.0707846  8.9390 < 2.2e-16 ***
mu   27.8541667  2.9510198  9.4388 < 2.2e-16 ***
prob  0.0222118  0.0033463  6.6378 3.184e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
fit_data_count$llr
```

```
Likelihood ratio test

               LogLik Df  Chisq Pr(>Chisq)
random :     -2654.71
aggregated :  -616.51  1 4076.4  < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The very low value of the P-value of the LLR test suggest that the negative binomial provides a better fit to the data. The `plot()` function allows for visualizing the expected random and aggregated frequencies together with the observed frequencies. The number of breaks can be adjusted as indicated.

```
plot(fit_data_count, breaks = 5)
```

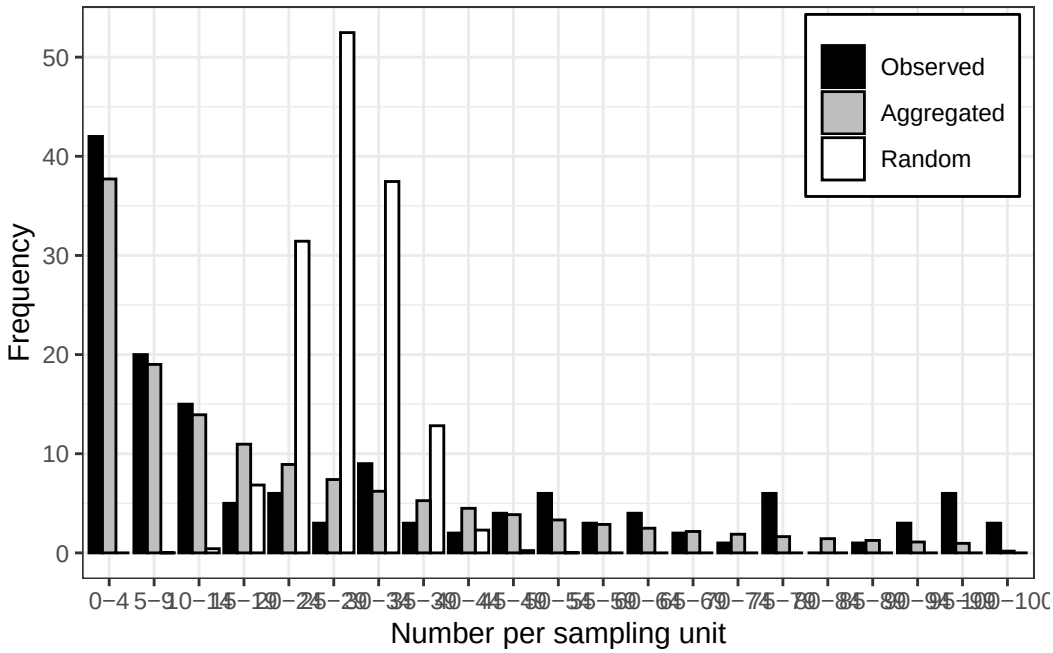


Figure 17.15: Frequencies of the observed and expected aggregated and random distributions

See below another way to plot by extracting the frequency data (and pivoting from wide to long format) from the generated list and using *ggplot*. Clearly, the negative binomial is a better description for the observed count data.

```
df <- fit_data_count$freq |>
  pivot_longer(2:4, names_to = "pattern", values_to = "value")

df |>
  ggplot(aes(category, value, fill = pattern)) +
  geom_col(position = "dodge", width = 2) +
  scale_fill_manual(values = c("gray70", "darkred", "steelblue")) +
  theme_r4pde()+
  theme(legend.position = "top")
```

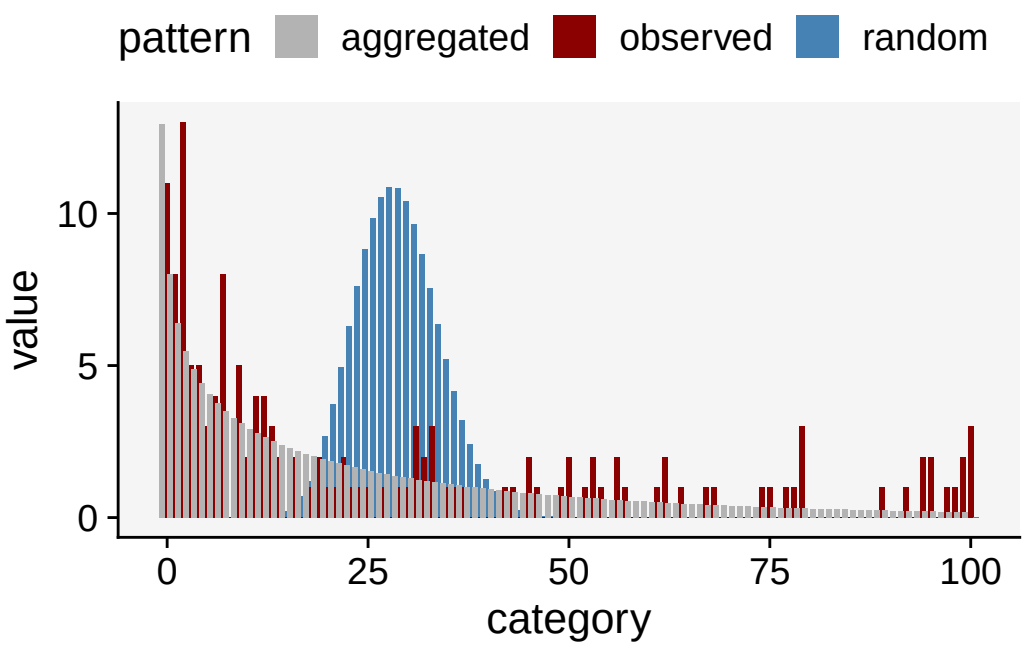


Figure 17.16: Frequencies of the observed and expected aggregated and random distributions

17.2.1.2 Aggregation indices

```
idx <- agg_index(data_count, method = "fisher")
idx
```

```
Fisher's index of dispersion:
(Version for count data)
34.25
```

```
chisq.test(idx)
```

```
    Chi-squared test for (N - 1)*index following a chi-squared
    distribution (df = N - 1)

data:  idx
X-squared = 4897.2, df = 143, p-value < 2.2e-16
```

```
z.test(idx)
```

```
    One-sample z-test

data:  idx
z = 82.085, p-value < 2.2e-16
alternative hypothesis: two.sided
```

```
# Lloyd index

idx_lloyd <- agg_index(data_count, method = "lloyd")
idx_lloyd
```

```
Lloyd's index of patchiness:
2.194
```

```
idx_mori <- agg_index(data_count, method = "morisita")
idx_mori
```

```
Morisita's coefficient of dispersion:
(Version for count data)
2.186
```

```
# Using the vegan package
library(vegan)
z <- data_count$data$i
mor <- dispindmorisita(z)
mor
```

```
      imor     mclu      muni      imst pchisq
1 2.185591 1.008728 0.9922162 0.5041152      0
```

17.2.1.3 Power law

When we have a collection of count data sets at the sampling unit scale the Taylor's power law (TPL) can be used to assess the overall degree of heterogeneity.

17.2.2 Incidence data

17.2.2.1 Fit to distributions

```
tas <-
  read.csv(
    "https://www.apsnet.org/edcenter/disimpactmngmnt/topc/EcologyAndEpidemiologyInR/Spatia
    sep = ""
  )
head(tas,10)
```

```
   quad group_size count
1     1          6     4
2     2          6     6
3     3          6     6
4     4          6     6
5     5          6     6
6     6          6     6
7     7          6     6
8     8          6     6
9     9          6     4
10   10          6     6
```

```
# Create incidence object for epiphy
dat_tas <- tas |>
  mutate(n = group_size, i = count) |>
  epiphy::incidence()

## Fit to two distributions
fit_tas <- fit_two_distr(dat_tas)
summary(fit_tas)
```

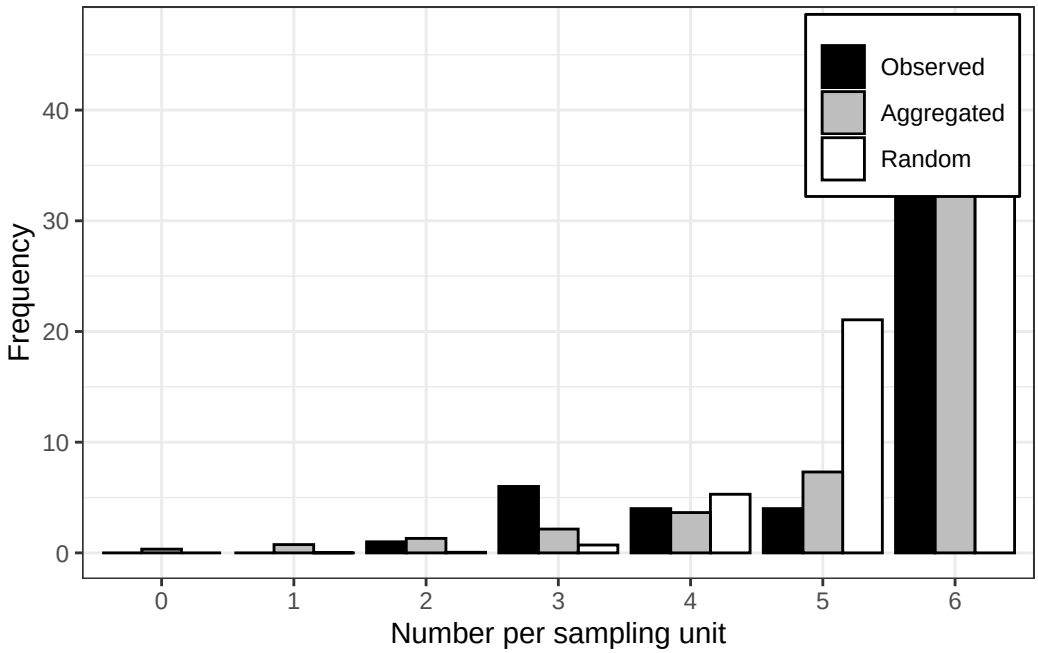
```
Fitting of two distributions by maximum likelihood
for 'incidence' data.
Parameter estimates:

(1) Binomial (random):
     Estimate Std.Err Z value    Pr(>z)
prob  0.90860 0.01494  60.819 < 2.2e-16 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(2) Beta-binomial (aggregated):
      Estimate  Std.Err Z value    Pr(>z)
alpha 1.923479 0.869621  2.2119  0.026976 *
beta  0.181337 0.075641  2.3973  0.016514 *
prob  0.913847 0.023139 39.4943 < 2.2e-16 ***
rho   0.322080 0.096414  3.3406  0.000836 ***
theta 0.475101 0.209789  2.2647  0.023534 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
fit_tas$llr
```

```
Likelihood ratio test

              LogLik Df  Chisq Pr(>Chisq)
random :     -75.061
aggregated : -57.430  1 35.263   2.88e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
plot(fit_tas)
```



17.2.2.2 Aggregation indices

glm model

```
binom.tas = glm(cbind(count, group_size - count) ~ 1,
                family = binomial,
                data = tas)
summary(binom.tas)
```

```
Call:
glm(formula = cbind(count, group_size - count) ~ 1, family = binomial,
    data = tas)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-3.447   1.073   1.073   1.073   1.073

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   2.2967     0.1799   12.77   <2e-16 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 117.76  on 61  degrees of freedom
Residual deviance: 117.76  on 61  degrees of freedom
AIC: 152.12

Number of Fisher Scoring iterations: 5
```

```
library(performance)
check_overdispersion(binom.tas)
```

```
# Overdispersion test

       dispersion ratio =   2.348
  Pearson's Chi-Squared = 143.206
              p-value = < 0.001
```

epiphy(c-alpha test)

```
library(epiphy)
tas2 <- tas |>
  mutate(i = count,
         n = group_size) |>  # create i vector
  epiphy::incidence()

t <- agg_index(tas2, flavor = "incidence")
t
```

```
Fisher's index of dispersion:
(Version for incidence data)
2.348
```

```
calpha.test(t)
```

```
    C(alpha) test
```

```
data:  t
z = 7.9886, p-value = 1.365e-15
```

17.2.2.3 Binary power law

When dealing with incidence data sets at the sampling unit scale, one often aims to discern patterns or tendencies in the data, especially with regards to the distribution of diseased individuals. The binary form of the power law is a statistical method used to assess the overall degree of heterogeneity in such data sets. In essence, this method compares the observed variance in the number of diseased individuals in the data set to the variance one would expect under a random distribution.

For incidence data, which deals with proportions, the expected random distribution is typically the Binomial distribution. The Binomial distribution is a discrete probability distribution that describes the number of successes in a fixed number of binary experiments. In the context of disease incidence, a "success" can be interpreted as the occurrence of a disease in a given sampling unit.

By comparing the observed variance to the variance under the Binomial distribution, researchers can determine if the disease is spreading in a manner that is consistent with random chance, or if there are significant deviations that might suggest other underlying factors at play.

Part IV

Epidemics and yield

18 Definitions and concepts

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

18.1 Introduction

Plant disease epidemics significantly impact agricultural production, particularly affecting crop yield - the measurable produce such as seed, fruit, leaves, roots or tubers - and quality, which includes factors such as blemishes on fruit and toxins in grain. Studying these impacts is crucial to understanding the overall repercussions of plant diseases on agriculture.

The yield of some crops can be severely diminished if they host a pathogen for a prolonged period of time. The plant's physiology is dynamically and negatively affected as the crop grows, leading to an increase in biomass and advancement through phenological stages. For some diseases that primarily infect the end product, like grains, yield is directly impacted with a reduction in size and weight of the affected plant part.

Certain plant diseases cause visual damage to the product, such as fruit or tubers, which may not result in a reduction in yield, but the presence of the symptoms can adversely affect sales due to decreased marketability. Furthermore, the presence of toxins in the product caused by some diseases can significantly downgrade its value, posing both health risks and economic losses.

Losses due to plant diseases can be categorized as direct, affecting the farm itself, or indirect, having broader impacts on society. Direct losses on the farm due to plant diseases are primarily due to reductions in the quantity and quality of yield, as well as the costs associated with disease control. These are classified as primary losses.

Secondary losses on the farm are indirect consequences of disease epidemics, such as the buildup of inoculum in the soil, which can lead to subsequent disease outbreaks. Other secondary impacts include the reduced efficacy of disease control methods due to the emergence of resistance to chemicals within the pathogen population over time.

In addition to these on-farm losses, plant diseases can have significant indirect impacts on society. These can include increased food prices due to reduced supply, loss of export markets due to trade restrictions, and environmental damage due to increased use of pesticides.

Understanding the full spectrum of losses caused by plant diseases is critical for developing effective disease management strategies and policies.

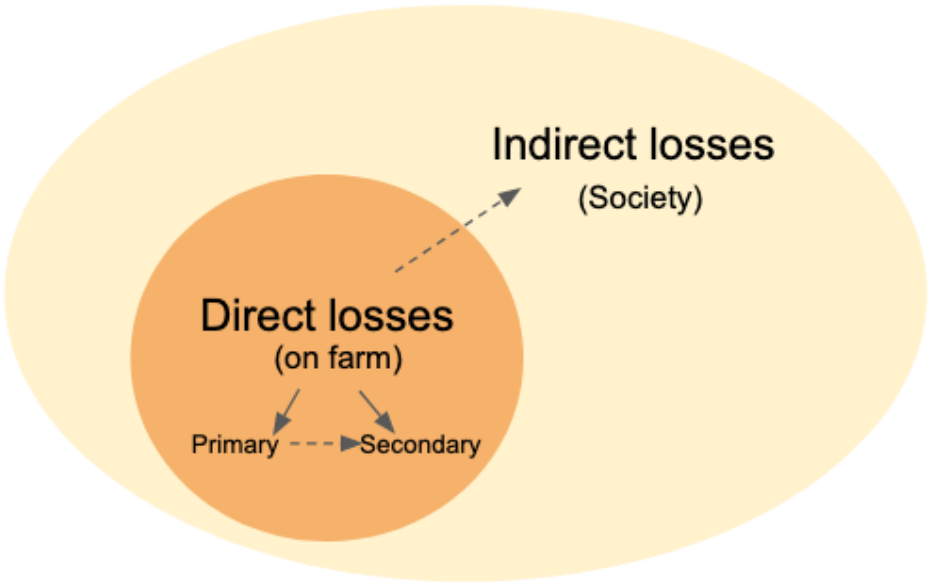


Figure 18.1: Tipology of losses caused by plant diseases

The famous epidemics in the ancient history, such as the late blight of potatoes, serve us as a remind of worst-case scenarios of major impact of epidemics causing both direct and indirect losses. However, crop losses due to diseases occur regularly and at levels that depend on the intensity of epidemics (Madden et al. 2007a). Expert opinion estimates have indicated that around 20% (on average) of the yield of major crops like wheat, rice, maize, potato and soybean is lost due to the pests and pathogens globally (Savary et al. 2019).

18.2 Crop loss assessment

According to Madden et al. (2007a), knowledge about the disease:yield relationship falls within crop loss assessment, a general branch of epidemiology that study the relationship between the attack by harmful organisms and the resulting yield (or yield loss) of crops. In fact, the study (analysis and modeling) of crop losses is considered central to plant pathology as no plant protection scientific reasoning could be possible without a measure of crop loss (Savary et al. 2006).

The concept of yield levels is important to recognize as a framework to study crop losses. There are three levels (from higher to lower) of yield: theoretical, attainable and actual.

- **Theoretical** (also known as potential) yield is determined mainly by *defining factors* such as the genotype of the crop grown under ideal conditions. It can be obtained in experimental plots managed with high input of fertilizers and pesticides.
- **Attainable** yield is obtained in commercial crops managed with a full range of modern technology to maximize yield. It considers the presence of *limiting factors* such as water and fertilizers.
- **Actual** yield is generally less than or equal to attainable yield, and is obtained under the effect of *reducing factors* such as those caused by pest (disease, insects, weeds) injuries - defined as measurable symptom caused by a harmful organism. It is the crop yield actually harvested in a farmer's field.

Yield loss (expressed in absolute or relative terms) is the difference between the attainable and the actual yield. Yield loss studies are only possible when reliable field data are collected in sufficient number to allow the development of statistical (empirical) models as well as the validation of mechanistic simulation yield loss models.

```
library(tidyverse)
yl <- tibble::tribble(
  ~yield, ~value, ~class,
  "Theoretical", 25,1,
  "Attainable", 20,2,
  "Actual", 15,3,
  "", 0, 4
)
yl |>
  ggplot(aes(reorder(yield,-value), value, fill = class))+
  geom_col(width = 0.5)+
  theme_grey(base_size = 16)+
  ggthemes::scale_fill_gradient_tableau(palette = "Green")+
  geom_hline(yintercept = 25, linetype = 2, color = "gray60")+
  geom_hline(yintercept = 20, linetype = 2, color = "gray60")+
  geom_hline(yintercept = 15, linetype = 2, color = "gray60")+
  theme(legend.position = "none",
        axis.text.y=element_blank(),
        axis.ticks.x = element_blank())+
  annotate(geom = "text", x = 1.8, y = 22, label ="Defining factors
           (genotype and environment)")+
  annotate(geom = "text", x = 2.8, y = 17.5, label ="Limiting factors
        (fertilizers, water)")+
```

```
annotate(geom = "text", x = 3.8, y = 12, label ="Reducing factors
         (pest, weeds, diseases)")+
annotate("segment", x = 4, y = 20, xend = 4, yend = 15,
       arrow = arrow(type = "closed", length = unit(0.02, "npc")))+
annotate(geom = "text", x = 4.3, y = 17, label ="Yield loss")+
labs(x = "", y = "")
```

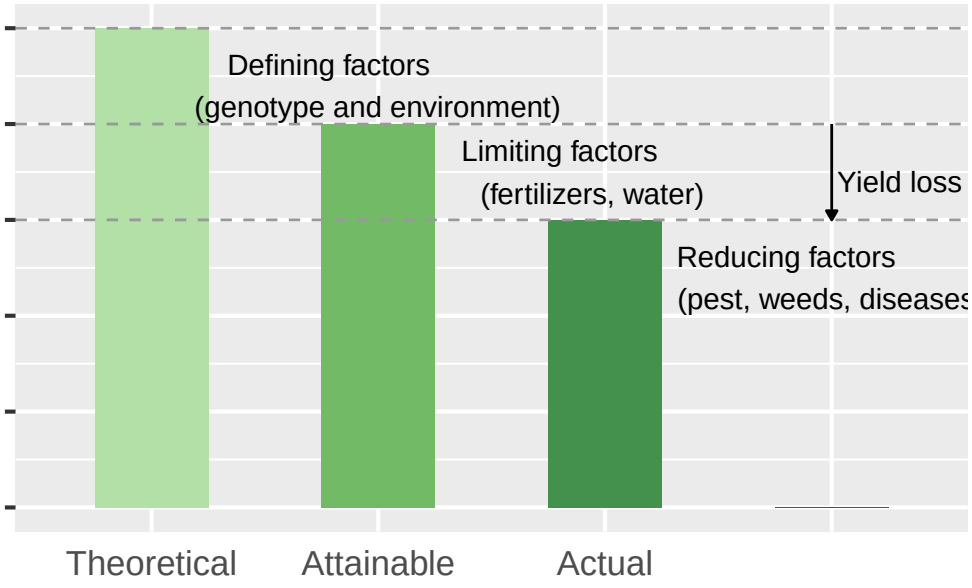


Figure 18.2: Yield levels

18.3 Disease:yield data and graphs

The datasets utilized to characterize a disease-yield relationship should ideally encompass a broad spectrum of yield and disease values. There are primarily two approaches to acquiring such data: 1) conducting experiments in controlled environments such as fields or greenhouses; or 2) conducting surveys in commercial fields that are naturally infected.

In experimental setups, researchers rely on different treatments that are designed to result in varying disease epidemics, under the assumption that the disease has an impact on yield. These treatments often include manipulating the level of inoculum when the disease is expected to be minimal. This is achieved through inoculations with different amounts of the pathogen. Conversely, when the disease is expected to be severe, researchers might use fungicides at different rates, frequencies, or timings.

An alternative strategy is to use different host genotypes, preferably isolines or near-isolines, which exhibit varying degrees of susceptibility to the disease. Another method is to manipulate the environment, for example by altering the irrigation levels.

In any case, the relationship between a measure of yield (either absolute or relative) and the disease variable can be evaluated using scatter plots that depict a "damage curve" (Madden et al. 2007a). The disease variable most commonly represents the assessment of the disease at a singular critical point. However, sometimes data obtained from multiple assessments throughout the disease epidemic is used to calculate the area under the disease progress curve, which is then used to represent the disease variable. This offers a more comprehensive view of the disease's impact over time, and can better capture the complex relationships between disease progression and yield loss.

Let's work with actual data on the incidence of white mold disease and yield of soybean determined across different locations and years in Brazil (Lehner et al. 2016). The variation in disease and yield was obtained by applying different fungicides that varied in efficacy, thus resulting in variable final disease incidence. The data was made freely available in this repository and was included the the package that accompanies the book.

```
library(tidyverse)
library(r4pde)
wm <- WhiteMoldSoybean

glimpse(wm)
```

```
Rows: 382
Columns: 17
$ study           <dbl> 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 18, 18, 18, 18, 18~
$ treat           <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, ~
$ season          <chr> "2009/2010", "2009/2010", "2009/2010", "2009/2010", "2~
$ harvest_year    <dbl> 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, ~
$ location        <chr> "Agua Fria", "Agua Fria", "Agua Fria", "Agua Fria", "A~
$ state           <chr> "GO", "GO", "GO", "GO", "GO", "GO", "GO", "GO", "GO", ~
$ country         <chr> "Brazil", "Brazil", "Brazil", "Brazil", "Brazil", "Bra~
$ elevation       <dbl> 891, 891, 891, 891, 891, 891, 891, 891, 891, 891, 891,~
$ region          <chr> "Northern", "Northern", "Northern", "Northern", "North~
$ elevation_class <chr> "low", "low", "low", "low", "low", "low", "low", "low"~
$ inc_check       <dbl> 37.7, 37.7, 37.7, 37.7, 37.7, 37.7, 37.7, 37.7, 37.7, ~
$ inc_class       <chr> "low", "low", "low", "low", "low", "low", "low", "low"~
$ yld_check       <dbl> 3729, 3729, 3729, 3729, 3729, 3729, 3729, 3729, 3729, ~
$ yld_class       <chr> "high", "high", "high", "high", "high", "high", "high"~
$ inc             <dbl> 37.7, 11.6, 33.5, 1.0, 5.6, 1.0, 3.7, 0.0, 1.1, 0.0, 3~
$ scl             <dbl> 5092, 6154, 200, 180, 1123, 641, 1203, 521, 20, 0, 847~
```

```
$ yld             <dbl> 3729, 3739, 3863, 3904, 4471, 4313, 4177, 4001, 4090, ~
```

As seen above using `glimpse()` function, the full data set has 17 variables. Let's reduce the data set to a few variables (study, inc and yld) and the trials number 1 to 4.

```
wm2 <- wm |>
  select(study, inc, yld) |>
  filter(study %in% c(1,2, 3, 4))
wm2
```

```
# A tibble: 52 x 3
   study   inc   yld
   <dbl> <dbl> <dbl>
 1     1    76  2265
 2     1    53  2618
 3     1    42  2554
 4     1    37  2632
 5     1    29  2820
 6     1    42  2799
 7     1    55  2503
 8     1    40  2967
 9     1    26  2965
10     1    18  3088
# i 42 more rows
```

We can now produce the damage curves for each study. As it can be seen, the relationship can be adequately described by a straight line.

```
wm2 |>
  ggplot(aes(inc, yld,
             group = study,
             color = factor(study)))+
  geom_point(size = 2)+
   theme_grey(base_size = 16)+
  ggthemes::scale_color_colorblind()+
  geom_smooth(method = "lm", se = F, color = "black", fullrange = T)+
  ylim(1800, 3500)+
  facet_wrap(~study)+
  labs(x = "White mold incidence (%)",
       y = "Soybean yield (kg/ha)",
       color = "Study")
```

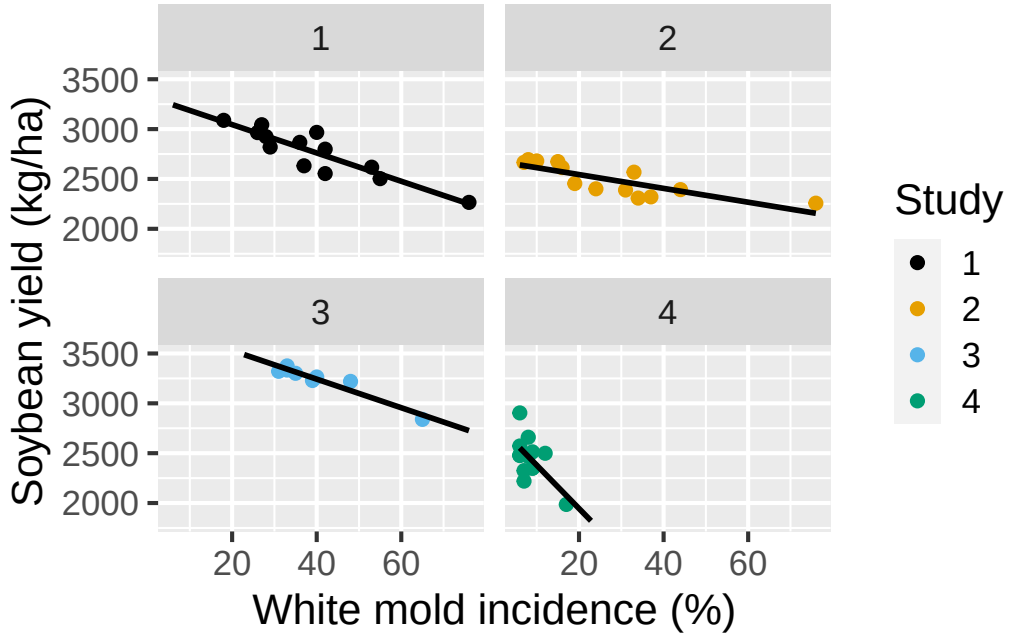


Figure 18.3: Relationship between soybean yield and incidence of white mold in two experiments

We can plot the relationships for all studies combined, which will resemble a "spaghetti" plot after adding the individual regression lines.

```
p1 <- wm |>
  ggplot(aes(inc, yld, group = study))+
  theme_grey(base_size = 16)+
  geom_point(size = 2, alpha = 0.5)+
  ylim(0, 5000)+
  labs(x = "White mold incidence (%)",
       y = "Soybean yield (kg/ha)")

p2 <- wm |>
  ggplot(aes(inc, yld, group = study))+
  theme_grey(base_size = 16)+
  geom_smooth(method = "lm", se = F, fullrange = T, color = "black")+
  ylim(0, 5000)+
  labs(x = "White mold incidence (%)",
       y = "Soybean yield (kg/ha)")
```

```
library(patchwork)
p1 | p2
```

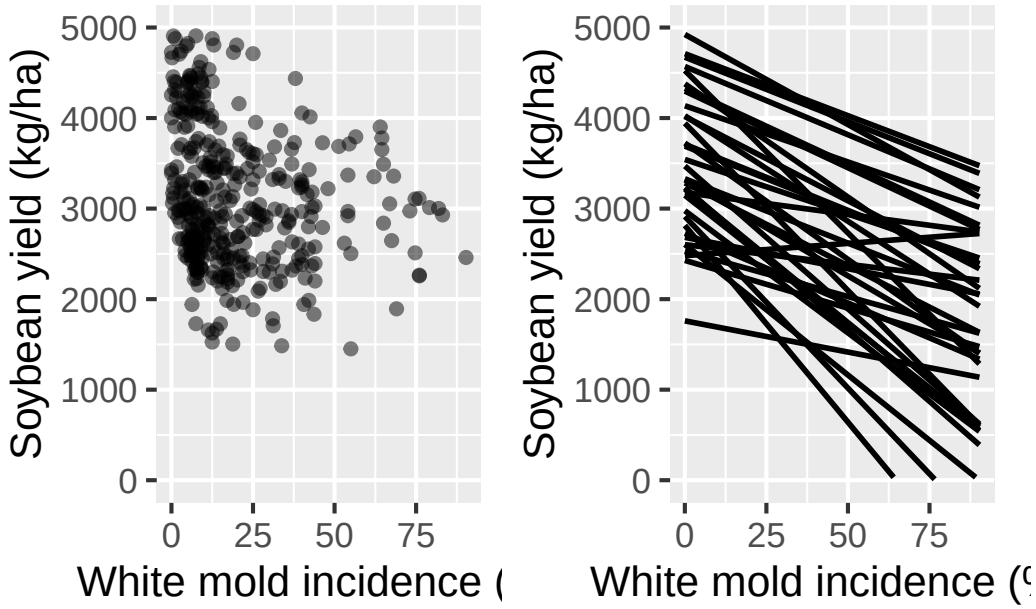


Figure 18.4: Relationship between soybean yield and incidence of white mold across trials. Observed (left) and fitted regression lines (right)

19 Statistical models

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

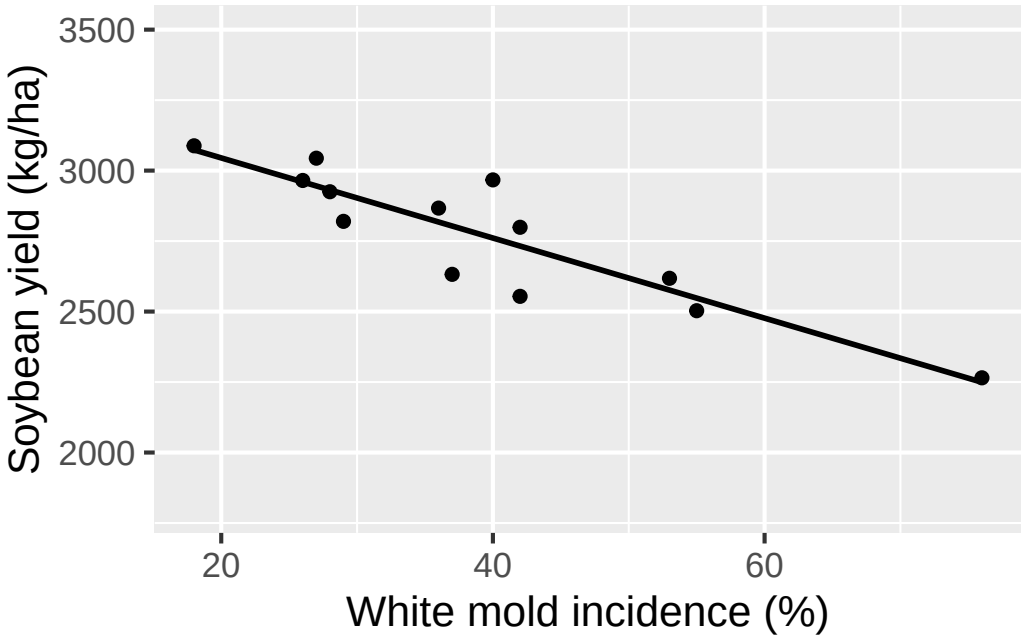
```
library(tidyverse)
theme_set(theme_bw(base_size = 16))
wm <- read_csv("https://raw.githubusercontent.com/emdelponte/paper-white-mold-meta-analysi
```

```
wm1 <- wm |>
  select(study, inc, yld) |>
  filter(study %in% c(1))
knitr::kable(wm1)
```

study	inc	yld
1	76	2265
1	53	2618
1	42	2554
1	37	2632
1	29	2820
1	42	2799
1	55	2503
1	40	2967
1	26	2965
1	18	3088
1	27	3044
1	28	2925
1	36	2867

```
wm1 |>
  ggplot(aes(inc, yld))+
  theme_grey(base_size = 16)+
  geom_point(size = 2)+
```

```
  geom_smooth(method = "lm", se = F, color = "black", fullrange = T)+
  ylim(1800, 3500)+
  labs(x = "White mold incidence (%)",
       y = "Soybean yield (kg/ha)",
       color = "Study")
```



Fitting a linear regression model

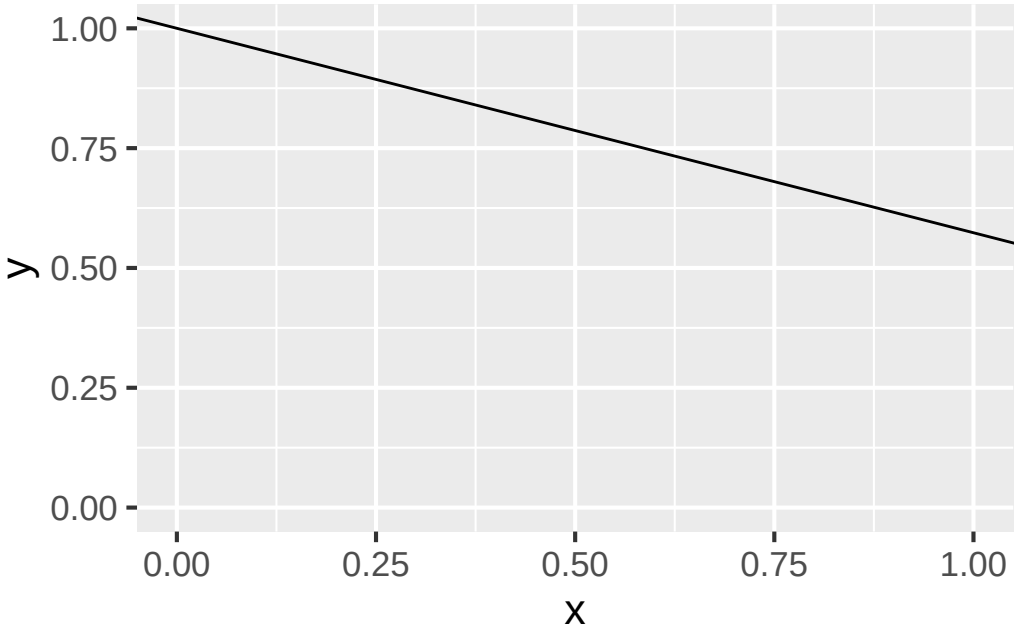
```
lm1 <-  lm(yld ~ inc, data = wm1)
summary(lm1)
```

```
Call:
lm(formula = yld ~ inc, data = wm1)

Residuals:
    Min      1Q  Median      3Q     Max
-178.41  -44.70   14.60   49.34  206.18

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 3329.142     86.844  38.335 4.60e-13 ***
```

```
inc          -14.208      2.076  -6.845 2.78e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 110.4 on 11 degrees of freedom
Multiple R-squared:  0.8099,    Adjusted R-squared:  0.7926
F-statistic: 46.86 on 1 and 11 DF,  p-value: 2.782e-05
```

The damage curves can be expressed in relative terms. For this, we divide the slope by the intercept and multiply by 100.

```r
slope <- -14.2/3329.14*100
x = seq(0,1,0.1)
y = seq(0,1,0.1)
dat <- data.frame(x,y)
dat |>
  ggplot(aes(x,y))+
  theme_grey(base_size = 16)+
  geom_point(color = "NA")+
  geom_abline(aes(intercept = 1, slope = slope))
```



References

Agrios, G. N. 2005a. INTRODUCTION. In Elsevier, p. 3–75. Available at: http://dx.doi.org/10.1016/b978-0-08-047378-9.50007-5.

Agrios, G. N. 2005b. Plant disease epidemiology. In Elsevier, p. 265–291. Available at: http://dx.doi.org/10.1016/b978-0-08-047378-9.50014-2.

Alves, K. S., and Del Ponte, E. M. 2021. Analysis and simulation of plant disease progress curves in R: introducing the epifitter package. Phytopathology Research. 3 Available at: http://dx.doi.org/10.1186/s42483-021-00098-7.

Alves, K. S., Guimarães, M., Ascari, J. P., Queiroz, M. F., Alfenas, R. F., Mizubuti, E. S. G., et al. 2021. RGB-based phenotyping of foliar disease severity under controlled conditions. Tropical Plant Pathology. 47:105–117 Available at: http://dx.doi.org/10.1007/S40858-021-00448-Y.

Baddeley, A., Diggle, P. J., Hardegen, A., Lawrence, T., Milne, R. K., and Nair, G. 2014. On tests of spatial pattern based on simulation envelopes. Ecological Monographs. 84:477–489 Available at: http://dx.doi.org/10.1890/13-2042.1.

Baddeley, A., Turner, R., Moller, J., and Hazelton, M. 2005. Residual analysis for spatial point processes (with discussion). Journal of the Royal Statistical Society: Series B (Statistical Methodology). 67:617–666 Available at: http://dx.doi.org/10.1111/j.1467-9868.2005.00519.x.

Barnhart, H. X., Haber, M., and Song, J. 2002. Overall Concordance Correlation Coefficient for Evaluating Agreement Among Multiple Observers. Biometrics. 58:1020–1027 Available at: http://dx.doi.org/10.1111/j.0006-341x.2002.01020.x.

Bock, C. H., Chiang, K.-S., and Del Ponte, E. M. 2021a. Plant disease severity estimated visually: a century of research, best practices, and opportunities for improving methods and practices to maximize accuracy. Tropical Plant Pathology. 47:25–42 Available at: http://dx.doi.org/10.1007/s40858-021-00439-z.

Bock, C. H., Pethybridge, S. J., Barbedo, J. G. A., Esker, P. D., Mahlein, A.-K., and Del Ponte, E. M. 2021b. A phytopathometry glossary for the twenty-first century: towards consistency and precision in intra- and inter-disciplinary dialogues. Tropical Plant Pathology. 47:14–24 Available at: http://dx.doi.org/10.1007/s40858-021-00454-0.

Brooks, M. E., Kristensen, K., van Benthem, K. J., Magnusson, A., Berg, C. W., Nielsen, A., et al. 2017. glmmTMB balances speed and flexibility among packages for zero-inflated generalized linear mixed modeling. The R Journal. 9:378–400.

Brown, V. A. 2021. An Introduction to Linear Mixed-Effects Modeling in R. Advances in Methods and Practices in Psychological Science. 4:251524592096035 Available at: http://dx.doi.org/10.1177/2515245920960351.

Café-Filho, A. C., Santos, G. R., and Laranjeira, F. F. 2010. Temporal and spatial dynamics of watermelon gummy stem blight epidemics. European Journal of Plant Pathology. 128:473–482 Available at: http://dx.doi.org/10.1007/s10658-010-9674-1.
Campbell, C. L., and Madden. L., V. 1990. *Introduction to plant disease epidemiology*. Wiley.
Chester, K. S. 1950. Plant disease losses : Their appraisal and interpretation /. Available at: http://dx.doi.org/10.5962/bhl.title.86198.
Chiang, K.-S., and Bock, C. H. 2021. Understanding the ramifications of quantitative ordinal scales on accuracy of estimates of disease severity and data analysis in plant pathology. Tropical Plant Pathology. 47:58–73 Available at: http://dx.doi.org/10.1007/s40858-021-00446-0.
Chiang, K.-S., Liu, S.-C., Bock, C. H., and Gottwald, T. R. 2014. What Interval Characteristics Make a Good Categorical Disease Assessment Scale? Phytopathology®. 104:575–585 Available at: http://dx.doi.org/10.1094/phyto-10-13-0279-r.
Cruz, C. D., and Valent, B. 2017. Wheat blast disease: danger on the move. Tropical Plant Pathology. 42:210–222 Available at: http://dx.doi.org/10.1007/s40858-017-0159-z.
Del Ponte, E. M., Cazón, L. I., Alves, K. S., Pethybridge, S. J., and Bock, C. H. 2022. How much do standard area diagrams improve accuracy of visual estimates of the percentage area diseased? A systematic review and meta-analysis. Tropical Plant Pathology. 47:43–57 Available at: http://dx.doi.org/10.1007/s40858-021-00479-5.
Del Ponte, E. M., Pethybridge, S. J., Bock, C. H., Michereff, S. J., Machado, F. J., and Spolti, P. 2017. Standard Area Diagrams for Aiding Severity Estimation: Scientometrics, Pathosystems, and Methodological Trends in the Last 25 Years. Phytopathology®. 107:1161–1174 Available at: http://dx.doi.org/10.1094/PHYTO-02-17-0069-FI.
Duffeck, M. R., Santos Alves, K. dos, Machado, F. J., Esker, P. D., and Del Ponte, E. M. 2020. Modeling Yield Losses and Fungicide Profitability for Managing Fusarium Head Blight in Brazilian Spring Wheat. Phytopathology®. 110:370–378 Available at: http://dx.doi.org/10.1094/PHYTO-04-19-0122-R.
Esser, D. S., Leveau, J. H. J., Meyer, K. M., and Wiegand, K. 2014. Spatial scales of interactions among bacteria and between bacteria and the leaf surface. FEMS Microbiology Ecology. 91 Available at: http://dx.doi.org/10.1093/femsec/fiu034.
Franceschi, V. T., Alves, K. S., Mazaro, S. M., Godoy, C. V., Duarte, H. S. S., and Del Ponte, E. M. 2020. A new standard area diagram set for assessment of severity of soybean rust improves accuracy of estimates and optimizes resource use. Plant Pathology. 69:495–505 Available at: http://dx.doi.org/10.1111/ppa.13148.
Francl, L. J. 2001. The..disease triangle: A plant pathological paradigm revisited. The Plant Health Instructor. Available at: http://dx.doi.org/10.1094/PHI-T-2001-0517-01.
Gigot, C. 2018. *Epiphy: Analysis of plant disease epidemics*.
Godoy, C. V., Seixas, C. D. S., Soares, R. M., Marcelino-Guimarães, F. C., Meyer, M. C., and Costamilan, L. M. 2016. Asian soybean rust in brazil: Past, present, and future. Pesquisa Agropecuária Brasileira. 51:407–421 Available at: http://dx.doi.org/10.1590/S0100-204X2016000500002.
González-Domínguez, E., Martins, R. B., Del Ponte, E. M., Michereff, S. J., García-Jiménez, J., and Armengol, J. 2014. Development and validation of a standard area diagram set to aid

assessment of severity of loquat scab on fruit. European Journal of Plant Pathology. Available at: http://dx.doi.org/10.1007/s10658-014-0400-2.
Hebert, T. T. 1982. The rationale for the horsfall-barratt plant disease assessment scale. Phytopathology. 72:1269 Available at: http://dx.doi.org/10.1094/phyto-72-1269.
Islam, M. T., Kim, K.-H., and Choi, J. 2019. Wheat Blast in Bangladesh: The Current Situation and Future Impacts. The Plant Pathology Journal. 35:1–10 Available at: http://dx.doi.org/10.5423/ppj.rw.08.2018.0168.
Jeger, M. J., and Viljanen-Rollinson, S. L. H. 2001. The use of the area under the disease-progress curve (AUDPC) to assess quantitative disease resistance in crop cultivars. Theoretical and Applied Genetics. 102:32–40 Available at: http://dx.doi.org/10.1007/s001220051615.
Jesus Junior, W. C. de, and Bassanezi, R. B. 2004. Análise da dinâmica e estrutura de focos da morte súbita dos citros. Fitopatologia Brasileira. 29:399–405 Available at: http://dx.doi.org/10.1590/S0100-41582004000400007.
Laranjeira, F. F., Bergamin Filho, A. R., and Amorim, L. I. 1998. Dinâmica e estrutura de focos da clorose variegada dos citros (CVC). Fitopatologia Brasileira. 23:36–41.
Laranjeira, F. F., Bergamin Filho, A., Amorim, L., and Gottwald, T. R. 2004. Dinâmica espacial da clorose variegada dos citros em três regiões do estado de são paulo. Fitopatologia Brasileira. 29:56–65 Available at: http://dx.doi.org/10.1590/S0100-41582004000100009.
Lehner, M. S., Pethybridge, S. J., Meyer, M. C., and Del Ponte, E. M. 2016. Meta-analytic modelling of the incidence–yield and incidence–sclerotial production relationships in soybean white mould epidemics. Plant Pathology. 66:460–468 Available at: http://dx.doi.org/10.1111/ppa.12590.
Li, B., Madden, L. V., and Xu, X. 2011. Spatial analysis by distance indices: an alternative local clustering index for studying spatial patterns. Methods in Ecology and Evolution. 3:368–377 Available at: http://dx.doi.org/10.1111/j.2041-210x.2011.00165.x.
Li, F., Upadhyaya, N. M., Sperschneider, J., Matny, O., Nguyen-Phuc, H., Mago, R., et al. 2019. Emergence of the Ug99 lineage of the wheat stem rust pathogen through somatic hybridisation. Nature Communications. 10 Available at: http://dx.doi.org/10.1038/s41467-019-12927-7.
Lin, L. I.-K. 1989. A concordance correlation coefficient to evaluate reproducibility. Biometrics. 45:255 Available at: http://dx.doi.org/10.2307/2532051.
Liu, H. I., Tsai, J. R., Chung, W. H., Bock, C. H., and Chiang, K. S. 2019. Effects of Quantitative Ordinal Scale Design on the Accuracy of Estimates of Mean Disease Severity. Agronomy. 9:565 Available at: http://dx.doi.org/10.3390/agronomy9090565.
Madden, L. V., Esker, P. D., and Pethybridge, S. J. 2021. Forrest W. Nutter, Jr.: a career in phytopathometry. Tropical Plant Pathology. 47:5–13 Available at: http://dx.doi.org/10.1007/s40858-021-00469-7.
Madden, L. V., Hughes, G., and Bosch, F. van den, eds. 2007a. CHAPTER 12: Epidemics and crop yield. In The American Phytopathological Society, p. 353–388. Available at: http://dx.doi.org/10.1094/9780890545058.012.
Madden, L. V., Hughes, G., and van den Bosch, F. 2007b. Spatial aspects of epidemics—III: Patterns of plant disease. In The American Phytopathological Society, p. 235–278. Available at: http://dx.doi.org/10.1094/9780890545058.009.

Madden, L. V., Hughes, G., and van den Bosch, F. 2007c. Temporal analysis i: Quantifying and comparing epidemics. In The American Phytopathological Society, p. 63–116. Available at: http://dx.doi.org/10.1094/9780890545058.004.
Madden, L. V., Hughes, G., and van den Bosch, F. 2007d. *The study of plant disease epidemics*. The American Phytopathological Society. Available at: http://dx.doi.org/10.1094/9780890545058.
Malaker, P. K., Barma, N. C. D., Tiwari, T. P., Collis, W. J., Duveiller, E., Singh, P. K., et al. 2016. First Report of Wheat Blast Caused by *Magnaporthe oryzae* Pathotype *triticum* in Bangladesh. Plant Disease. 100:2330–2330 Available at: http://dx.doi.org/10.1094/pdis-05-16-0666-pdn.
Mikaberidze, A., Mundt, C. C., and Bonhoeffer, S. 2015. Data from: Invasiveness of plant pathogens depends on the spatial scale of host distribution. Available at: http://datadryad.org/stash/dataset/doi:10.5061/dryad.f2j8s.
Moran, P. A. P. 1950. Notes on continuous stochastic phenomena. Biometrika. 37:17.
Moreira, R. R., Silva Silveira Duarte, H. da, and De Mio, L. L. M. 2018. Improving accuracy, precision and reliability of severity estimates of Glomerella leaf spot on apple leaves using a new standard area diagram set. European Journal of Plant Pathology. 153:975–982 Available at: http://dx.doi.org/10.1007/s10658-018-01610-0.
Mundt, C. C., Ahmed, H. U., Finckh, M. R., Nieva, L. P., and Alfonso, R. F. 1999. Primary Disease Gradients of Bacterial Blight of Rice. Phytopathology®. 89:64–67 Available at: http://dx.doi.org/10.1094/phyto.1999.89.1.64.
Nelson, S. C. 1996. A simple analysis of disease foci. Phytopathology. 86:432–439.
Nutter, F. W., and Esker, P. D. 2006. The Role of Psychophysics in Phytopathology: The Weber–Fechner Law Revisited. European Journal of Plant Pathology. 114:199–213 Available at: http://dx.doi.org/10.1007/s10658-005-4732-9.
Nutter, F. W., Esker, P. D., and Netto, R. A. C. 2006. Disease Assessment Concepts and the Advancements Made in Improving the Accuracy and Precision of Plant Disease Data. European Journal of Plant Pathology. 115:95–103 Available at: http://dx.doi.org/10.1007/s10658-005-1230-z.
Olivoto, T. 2022. Lights, camera, pliman! An R package for plant image analysis. Methods in Ecology and Evolution. 13:789–798 Available at: http://dx.doi.org/10.1111/2041-210X.13803.
Olivoto, T., Andrade, S. M. P., and M. Del Ponte, E. 2022. Measuring plant disease severity in R: introducing and evaluating the pliman package. Tropical Plant Pathology. 47:95–104 Available at: http://dx.doi.org/10.1007/s40858-021-00487-5.
Parker, S. K., Nutter, F. W., and Gleason, M. L. 1997. Directional Spread of Septoria Leaf Spot in Tomato Rows. Plant Disease. 81:272–276 Available at: http://dx.doi.org/10.1094/pdis.1997.81.3.272.
Pereira, W. E. L., Andrade, S. M. P. de, Del Ponte, E. M., Esteves, M. B., Canale, M. C., Takita, M. A., et al. 2020. Severity assessment in the Nicotiana tabacum-Xylella fastidiosa subsp. pauca pathosystem: design and interlaboratory validation of a standard area diagram set. Tropical Plant Pathology. 45:710–722 Available at: http://dx.doi.org/10.1007/s40858-020-00401-5.

Sackett, K. E., and Mundt, C. C. 2005. Primary Disease Gradients of Wheat Stripe Rust in Large Field Plots. Phytopathology®. 95:983–991 Available at: http://dx.doi.org/10.1094/PHYTO-95-0983.
Savary, S., Teng, P. S., Willocquet, L., and Nutter, F. W. 2006. Quantification and Modeling of Crop Losses: A Review of Purposes. Annual Review of Phytopathology. 44:89–112 Available at: http://dx.doi.org/10.1146/annurev.phyto.44.070505.143342.
Savary, S., Willocquet, L., Pethybridge, S. J., Esker, P., McRoberts, N., and Nelson, A. 2019. The global burden of pathogens and pests on major food crops. Nature Ecology & Evolution. 3:430–439 Available at: http://dx.doi.org/10.1038/s41559-018-0793-y.
Scott, P. R., and Hollins, T. W. 1974. Effects of eyespot on the yield of winter wheat. Annals of Applied Biology. 78:269–279 Available at: http://dx.doi.org/10.1111/j.1744-7348.1974.tb01506.x.
Shrout, P. E., and Fleiss, J. L. 1979. Intraclass correlations: Uses in assessing rater reliability. Psychological Bulletin. 86:420–428 Available at: http://dx.doi.org/10.1037/0033-2909.86.2.420.
Simko, I., and Piepho, H.-P. 2012. The Area Under the Disease Progress Stairs: Calculation, Advantage, and Application. Phytopathology®. 102:381–389 Available at: http://dx.doi.org/10.1094/phyto-07-11-0216.
Tembo, B., Mulenga, R. M., Sichilima, S., M'siska, K. K., Mwale, M., Chikoti, P. C., et al. 2020. Detection and characterization of fungus (Magnaporthe oryzae pathotype Triticum) causing wheat blast disease on rain-fed grown wheat (Triticum aestivum L.) in Zambia ed. Zonghua Wang. PLOS ONE. 15:e0238724 Available at: http://dx.doi.org/10.1371/journal.pone.0238724.
Thresh, J. M. 1998. In memory of James Edward Vanderplank 1909–1997. Plant Pathology. 47:114–115 Available at: http://dx.doi.org/10.1046/j.1365-3059.2998.00220.x.
Vanderplank, J. 1963. *Plant disease epidemics and control*. Elsevier. Available at: http://dx.doi.org/10.1016/C2013-0-11642-X.
Wiegand, T., and A. Moloney, K. 2004. Rings, circles, and null-models for point pattern analysis in ecology. Oikos. 104:209–229 Available at: http://dx.doi.org/10.1111/j.0030-1299.2004.12497.x.
Xu, X.-M., and Madden, L. V. 2004. Use of SADIE statistics to study spatial dynamics of plant disease epidemics. Plant Pathology. 53:38–49 Available at: http://dx.doi.org/10.1111/j.1365-3059.2004.00949.x.
Yadav, N. V. S., Vos, S. M. de, Bock, C. H., and Wood, B. W. 2012. Development and validation of standard area diagrams to aid assessment of pecan scab symptoms on fruit. Plant Pathology. 62:325–335 Available at: http://dx.doi.org/10.1111/j.1365-3059.2012.02641.x.
Yorinori, J. T., Paiva, W. M., Frederick, R. D., Costamilan, L. M., Bertagnolli, P. F., Hartman, G. E., et al. 2005. Epidemics of Soybean Rust (*Phakopsora pachyrhizi*) in Brazil and Paraguay from 2001 to 2003. Plant Disease. 89:675–677 Available at: http://dx.doi.org/10.1094/PD-89-0675.
Zadoks, J. C., and Schein, R. D. 1988. James Edward Vanderplank: Maverick* and Innovator. Annual Review of Phytopathology. 26:31–37 Available at: http://dx.doi.org/10.1146/annurev.py.26.090188.000335.