

R4PDE.net

R for Plant Disease Epidemiology

Visualize, Describe and Model
Epidemiological Data in R



Emerson M. Del Ponte

Contents

	Welcome	7
	How to cite	8
	About the author	10
	Acknowledgements	11
1	Introduction	12
1.1	Defining plant disease	12
1.2	Importance of epidemics	12
1.3	History of epidemiology	13
1.4	Other resources	15
2	Disease variables	17
2.1	Disease quantification	17
2.2	Disease intensity: incidence, prevalence and severity	18
2.3	Data types	20
2.4	Statistical distributions and simulation	21
3	Ordinal scales	30
3.1	Introduction	30
3.2	Disease severity index (DSI)	34
3.3	Analysis of ordinal data	37
4	Measuring foliar severity	45
4.1	The actual severity measure	45
4.2	Image palettes	45
4.3	Measuring severity	49
4.4	How good are these measurements?	57
5	Reliability and accuracy	60
5.1	Severity data	60
5.2	Incidence data	73
6	Standard area diagrams	76
6.1	Definitions	76
6.2	SAD development and validation	79
6.3	Designing SADs in R	80
6.4	Analysis of SADs validation data	81

7	Training sessions	91
7.1	Why train raters?	91
7.2	Software	91
8	Remote sensing	94
8.1	Introduction	94
8.2	Remote sensing background	94
8.3	Remote sensing of crop diseases	98
8.4	Disease detection	102
8.5	Disease quantification	124
8.6	Summary	155
9	Disease progress curves	159
9.1	How epidemics occur	159
9.2	Disease curves	161
9.3	Epidemic classification	162
9.4	Curve descriptors and AUDPC	164
9.5	Functional comparison of curves	167
10	Population models	172
10.1	Non-flexible models	172
10.2	Interactive application	180
11	Model fitting	181
11.1	Linear regression: single epidemics	181
11.2	Nonlinear regression	186
11.3	epifitter - multiple epidemics	188
11.4	Entering data	188
11.5	Visualize the DPCs	188
11.6	Designed experiments	201
12	Compartmental models	210
12.1	A starting point: the H-L-I-R model	211
12.2	Exploring the model through simulation	215
12.3	Key epidemiological quantities	218
12.4	Extending the H-L-I-R framework: adding biological realism	222
12.5	Fitting to data	232
12.6	Further extensions	238
12.7	Summary	239
12.8	Appendix. Relationship to population dynamics models	239
13	Spatial gradients	243
13.1	Introduction	243

14	Gradient models	250
14.1	Exponential model	250
14.2	Power law model	251
14.3	Linearization of the models	253
14.4	Interactive application	254
15	Fitting gradient models	256
15.1	Dataset	256
15.2	Visualize the gradient curve	256
15.3	Linear regression	257
15.4	Fitting all models with <code>fit_gradients()</code>	262
16	Spatial patterns	265
16.1	Definitions	265
16.2	Spatiotemporal	269
16.3	Simulating spatial patterns	270
17	Tests for patterns	272
17.1	Intensively mapped	272
17.2	Sparsely sampled data	309
18	Definitions and concepts	319
18.1	Introduction	319
18.2	Crop loss assessment	320
18.3	Generating disease:yield data	322
18.4	Damage curves	322
19	Statistical models	326
19.1	Introduction	326
19.2	Example 1: single study	326
19.3	Linear regression	327
19.4	Damage coefficients	328
19.5	Example 2: Multiple studies	330
20	Warning systems	339
20.1	Introduction	339
20.2	Risk assessment and decision framework	340
20.3	When is a warning system needed?	341
20.4	What types of systems are there?	342
20.5	Practical considerations	361
21	Disease modeling	363
21.1	Introduction	363

21.2	Controlled environment	365
21.3	Field data	387
	References	426

Welcome

R for Plant Disease Epidemiology (R4PDE) is an online book 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 and space. Readers should ideally have a foundational knowledge of R to best utilize the examples.

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. This book draws on multiple sources, but in some sections, it utilizes data and replicates (with permission) some of the analyses (presented in SAS code) from [The Study of Plant Disease Epidemics \(Madden et al., 2007a\)](#), a highly recommended textbook for anyone wishing to delve deeply into plant disease epidemiology.

Edition 2 (released 4 October 2026; edition 1 was first published on 23 October 2023). See [How to cite](#) for the DOI and how to cite this edition.

The chapters on remote sensing (Ivan A. Lizarazo) and on compartmental models (Nik Cuniffe) were written by guest authors; the rest of the book is by the author.

A mix of general and specific R packages is used 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 CRAN using:

```
install.packages("r4pde")
```

The development version can be installed from GitHub using `{pak}`:

```
install.packages("pak")
pak::pkg_install("emdelponte/r4pde")
```

This online book is frequently updated and edited. Its content is free to use, licensed under a [Creative Commons license](#), and the code for all analyses can be found on [GitHub](#). Contributions are subject to a [Contributor Code of Conduct](#), and by contributing, you agree to adhere to its terms.

[R for Plant Disease Epidemiology](#) © 2023–2026 by Emerson Medeiros Del Ponte is licensed under CC BY-NC 4.0

How to cite

The author has opted for self-publishing this book to ensure its accessibility and updatability, reflecting the dynamic nature of the field and the R programming environment. Readers are encouraged to cite this work when referencing or utilizing the methodologies and insights provided in the book. Ensure the accuracy of the details while citing, especially the URL, to direct readers to the correct online source.

This is the second edition of the book. It was first published on 23 October 2023, and edition 2 was released on 4 October 2026. The book is continuously updated: the DOI identifies the archived snapshot of the edition, and the website may be newer.

A suggested citation for the whole book (please use the edition and the year you consulted):

Del Ponte, E. M. (2026). *R for Plant Disease Epidemiology (R4PDE)* (2nd ed.). Author. DOI: [10.5281/zenodo.23145652](https://doi.org/10.5281/zenodo.23145652). <https://r4pde.net>

Edition 1 (first published in 2023) is archived on Zenodo (DOI: [10.5281/zenodo.23145275](https://doi.org/10.5281/zenodo.23145275)).

Two chapters were written by guest authors, and can also be cited individually:

Lizarazo, I. A. (2026). Remote sensing. In: Del Ponte, E. M., *R for Plant Disease Epidemiology (R4PDE)*. <https://r4pde.net/data-remote-sensing.html>

Cunniffe, N. J. (2026). Compartmental models. In: Del Ponte, E. M., *R for Plant Disease Epidemiology (R4PDE)*. <https://r4pde.net/temporal-compartmental.html>

Cataloguing-in-Publication data prepared by the library of the Federal University of Viçosa, Brazil.

**Ficha catalográfica elaborada pela Biblioteca Central da Universidade
Federal de Viçosa – Campus Viçosa**

D363r Del Ponte, Emerson Medeiros, 1973-
2023 R for plant disease epidemiology [recurso eletrônico] :
visualize, describe and model epidemiological data in R /
Emerson Medeiros Del Ponte -- Viçosa, MG : O Autor, 2023.
1 livro eletrônico.

Texto em inglês.

Disponível em: <https://r4pde.net>

1. Plantas – Doenças. 2. Epidemias. 3. Modelagem. 4.
Estatística. I. Título.

CDD 22. ed. 632

Bibliotecária responsável: Alice Regina Pinto Pires - CRB-6/2523

About the author

I was born in 1973 in Pelotas, in southern Brazil. My interest in agriculture began early, influenced by relatives and family friends who worked in agricultural sciences. My high school education in a rural school further strengthened this connection. I graduated in agronomy in 1996 and later pursued graduate studies in plant pathology at the Federal University of Pelotas. During my doctoral research, I worked on the epidemiology and modeling of Fusarium head blight, an important disease of wheat.

Part of my doctoral training was completed during a one-year visit to Cornell University. I later held a postdoctoral position at Iowa State University, where I worked on disease risk assessment and prediction. In 2006, I returned to Brazil as an assistant professor at the Federal University of Rio Grande do Sul, where I worked for eight years in teaching, research, and student supervision in plant disease epidemiology. In 2014, I joined the Universidade Federal de Viçosa (UFV), where I currently serve as full professor and supervise master's and doctoral students.

Open science has been a consistent part of my academic work. In [my lab](#), we use R for statistical analysis, data processing, and reproducible research. We also share scripts, preprints, and open-access outputs when possible.

This approach also shaped the development of *R for Plant Disease Epidemiology*, which was written with open-source tools and published in an open format. This allows the book to remain accessible and to be revised as methods, data, and software evolve.

I dedicate this book to my wife, Isabel, and our son, Vitor, for their support throughout the writing and revision of this book.

Acknowledgements

Writing *R for Plant Disease Epidemiology (R4PDE)* has benefited from the support and contributions of several people and organizations.

I thank Helen Pennington for granting permission to use her painting of coffee leaf rust on the book cover. Her work provides an appropriate visual representation of one of the classic subjects in plant disease epidemiology.

I am grateful to Prof. Laurence V. Madden for sharing data from *The Study of Plant Disease Epidemics*. These datasets contributed substantially to the analytical examples presented in the book.

I also thank the contributing chapter authors for sharing their expertise. Prof. Ivan A. Lizarazo contributed the chapter on remote sensing for disease detection and quantification. Prof. Nik Cunniffe contributed the chapter on compartmental models in plant disease epidemiology.

Finally, I acknowledge the open-source community for contributions that improved *R4PDE*, including code fixes, suggestions, pull requests, and other forms of feedback. Contributors include Adam Sparks (@adamhsparks), Remco Stam (@remco-stam), Tiago Olivoto (@TiagoOlivoto), Monalisa De Cól (@Monalisacdc), Juan Edwards Molina (@juanchiem), Onunkwor Raphael (@Onunkwor), Denis Shah (@DenisAShah), Ivan A. Lizarazo, and Juan A. Paredes.

1. Introduction

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 a 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., 2007a).

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 represents the evolution of a plant disease through time (Francl, 2001).

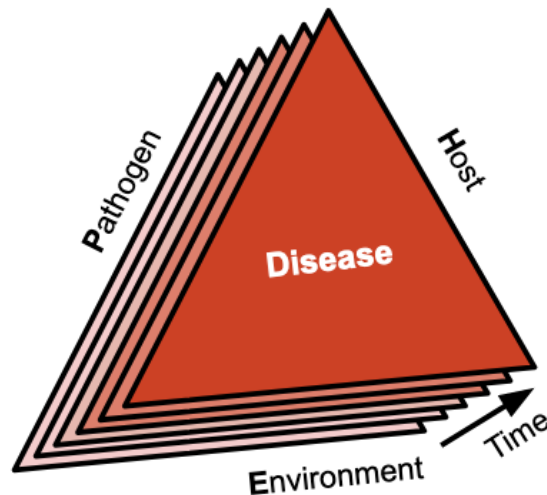


Figure 1.1: The plant disease prism as a model of plant disease epidemics

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, consider 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 losses 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 had been restricted to South America, was first detected in South Asia, Bangladesh, in 2016. Blast epidemics on that occasion devastated more than 15,000 ha of wheat and reduced yield of wheat in the affected field up to 100% (Islam et al., 2019; Malaker et al., 2016). 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 & Valent, 2017).

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 du 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 Kühn, 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 (Thresh, 1998; Zadoks & Schein, 1988), 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.



Figure 1.2: Group photo of the First International Epidemiology Workshop

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:

- 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

2018 - Lillehammer, Norway

2024 - Iguassu Falls, Brazil

Other resources

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

Online tutorials

Ecology and Epidemiology in R

Plant Disease Epidemiology - Temporal aspects

Simulation Modeling in Plant Disease Epidemiology and Crop Loss Analysis

Software

Epicrop - Simulation Modeling of Crop Diseases using a SEIR model

PART I

Epidemic data

2. Disease variables

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., 2021a).

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.

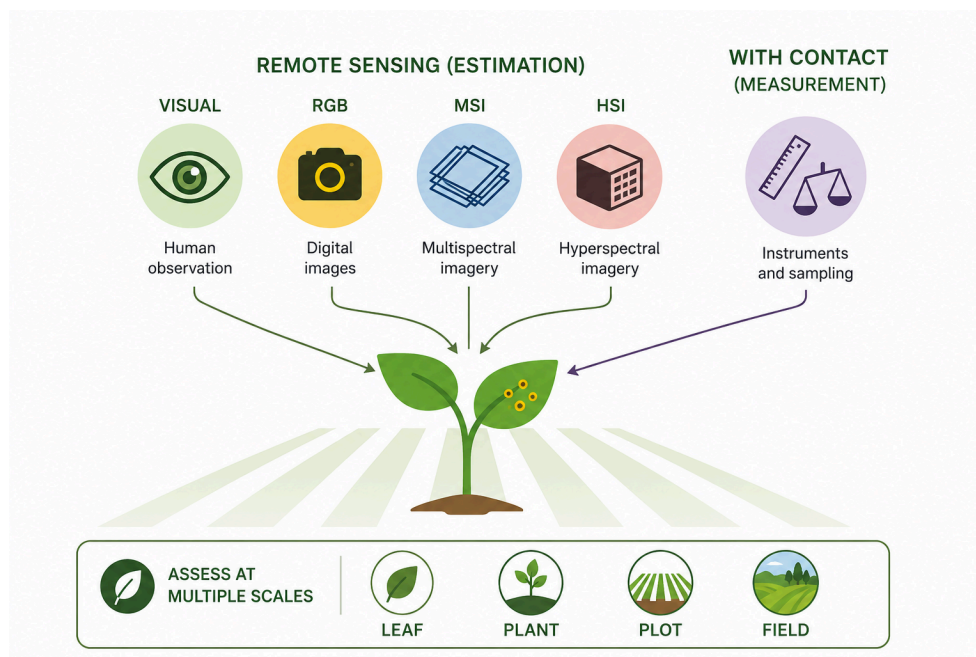


Figure 2.1: Different approaches used to obtain estimates or measures of plant disease. RGB = red, green, blue; MSI = multispectral imaging; HSI = hyperspectral imaging.

Finally, 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.

Disease intensity: incidence, prevalence and severity

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.

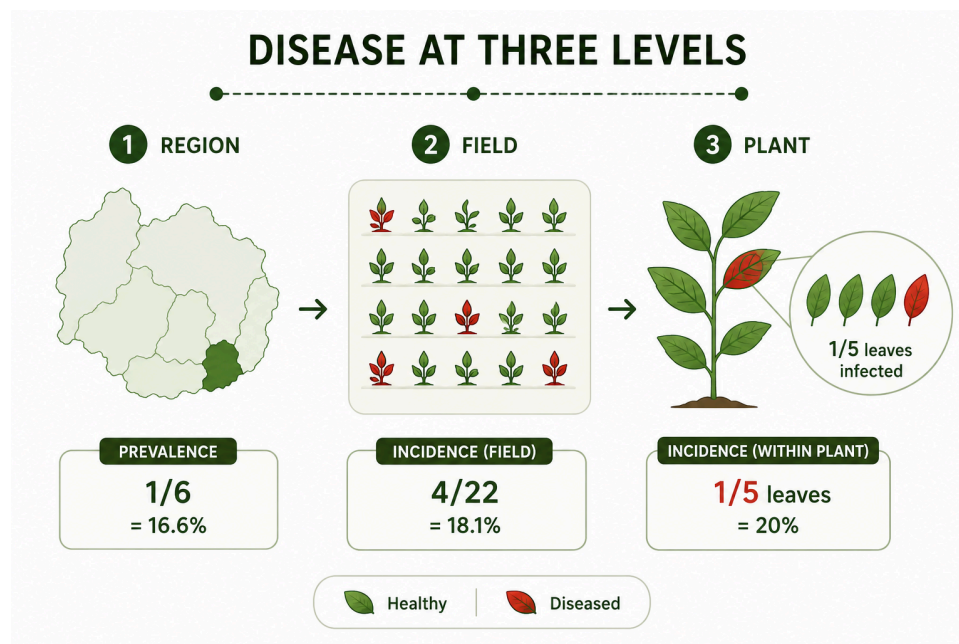


Figure 2.2: Schematic representation of how prevalence and incidence of plant diseases are calculated depending on the spatial scale of the assessment

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 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., 2021a). 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).

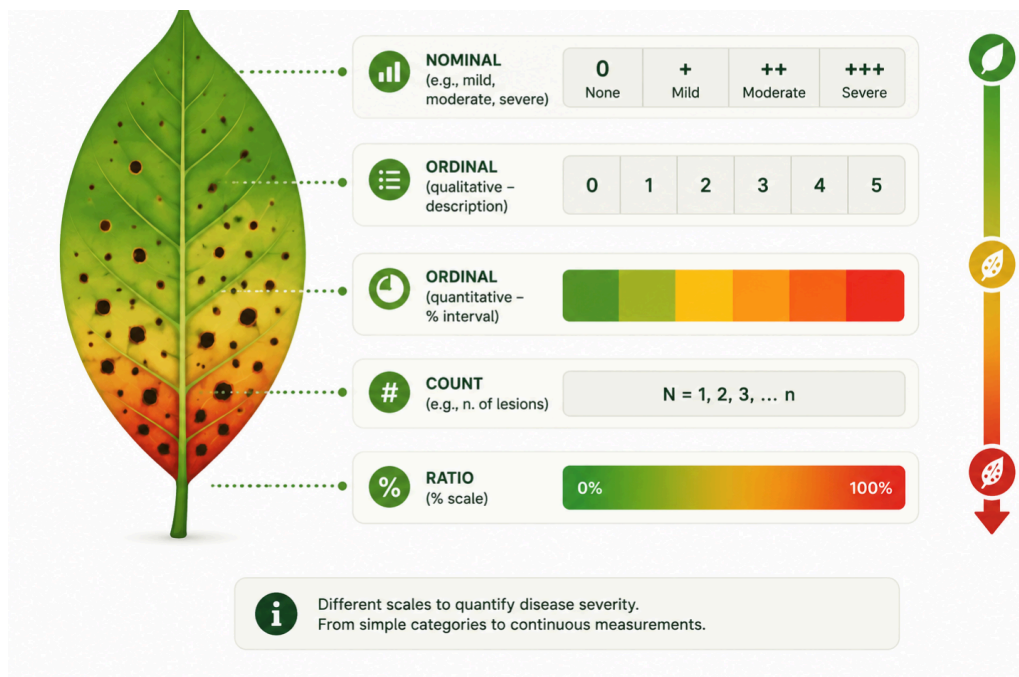


Figure 2.3: Scales and associated levels of measurement used to describe severity of plant diseases

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 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 is a model that provides 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.

Statistical distributions and simulation

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 diseased or not diseased. The statistical distribution that best describes 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()
```

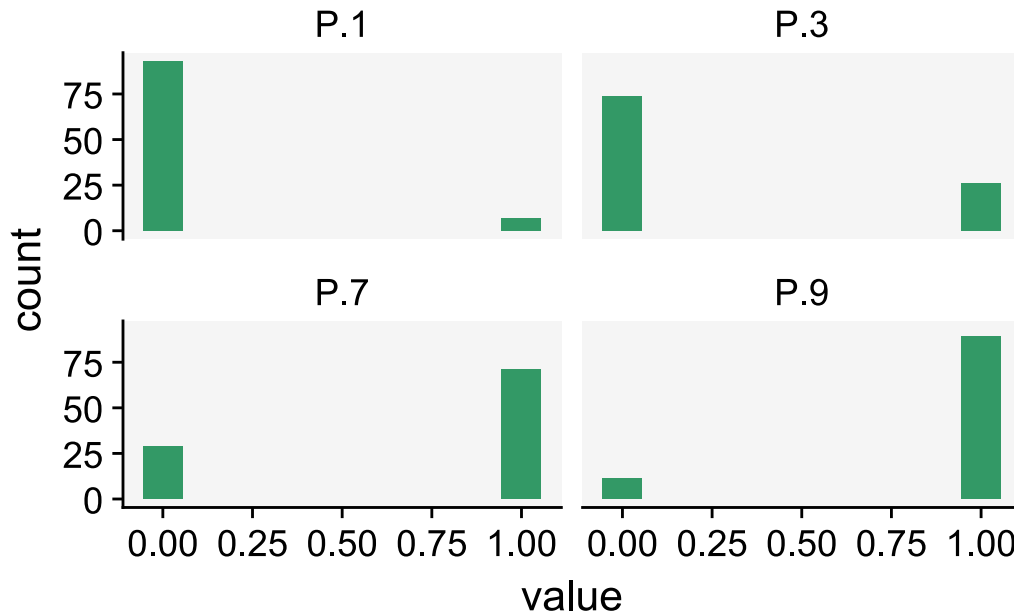


Figure 2.4: Binomial distribution to describe binary data

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.

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.

```
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()

```

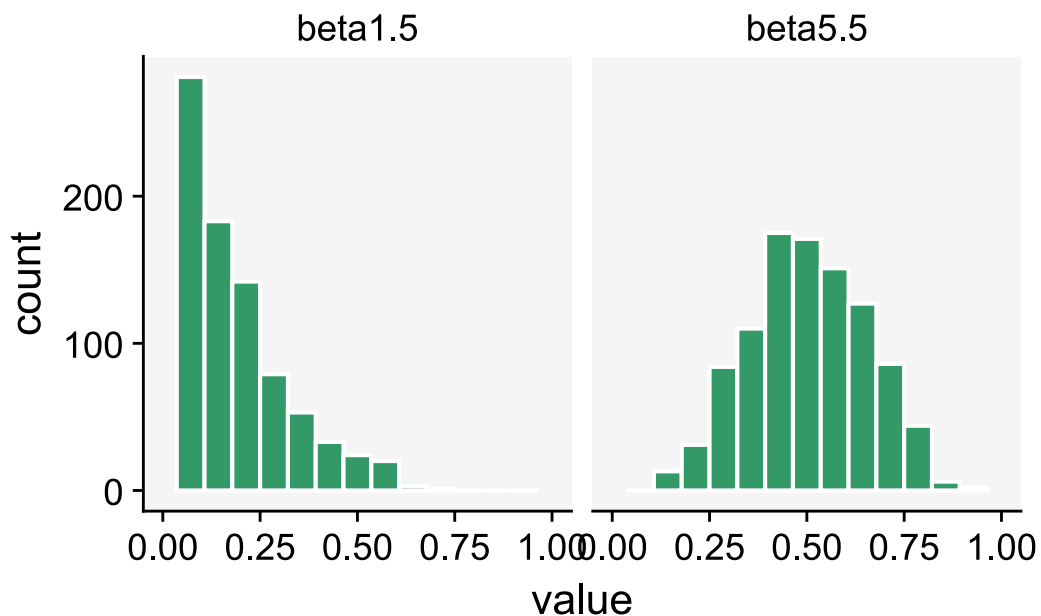


Figure 2.5: Beta distribution to describe proportion data

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 distribution'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 show more variance than the binomial 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 the 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()
```

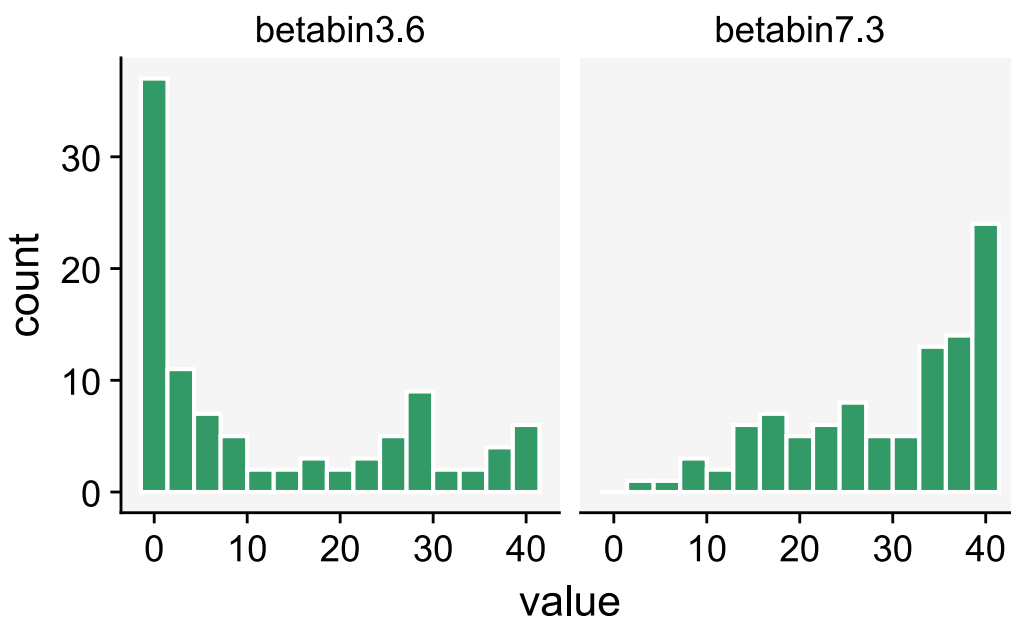


Figure 2.6: Beta-binomial distribution to describe proportion data

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 use the `rpois()` function to obtain 100 random observations following a Poisson distribution. To do so, we need to specify the number of observations ($n = 100$) and `lambda`, the vector of means.

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

```
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()
```

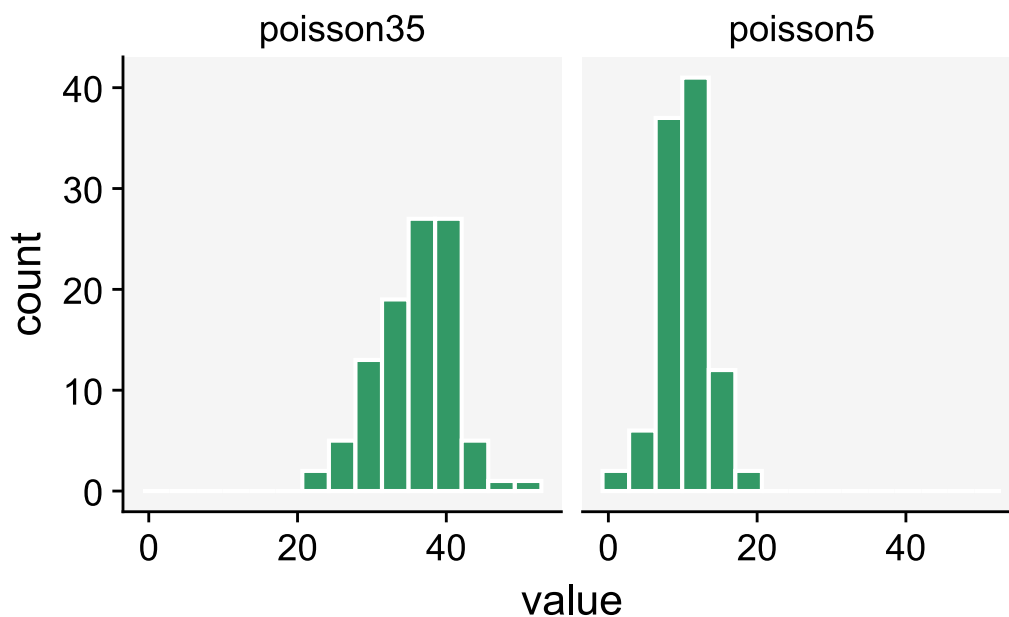


Figure 2.7: Poisson distribution to describe count data

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 failures in a

sequence of independent and identically distributed Bernoulli trials before a specified (non-random) number of successes 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 (λ , 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()
```

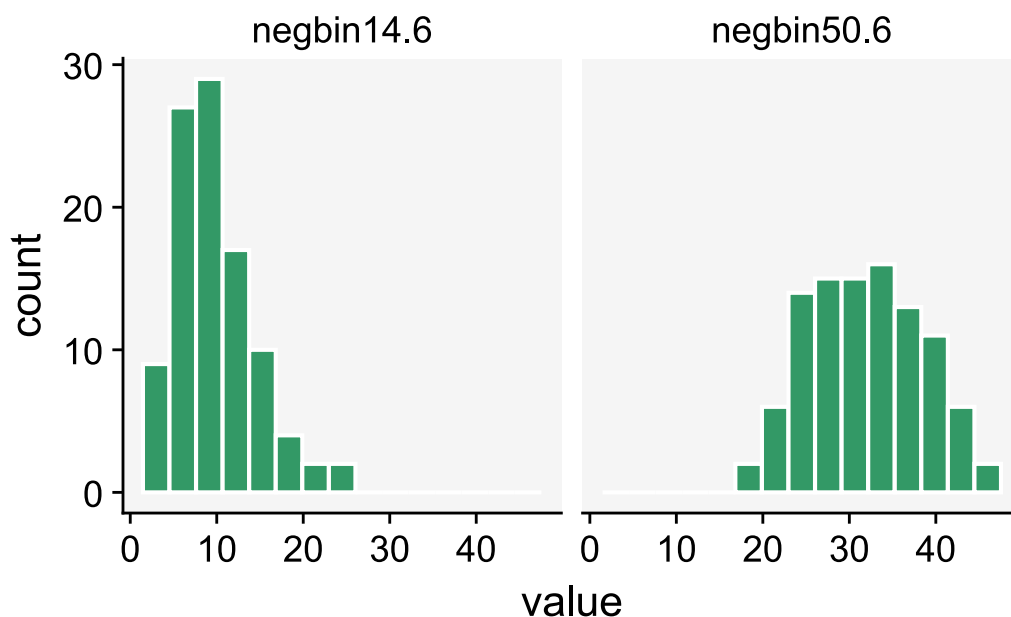


Figure 2.8: Negative binomial distribution to describe overdispersed count data

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 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` (with `scale = 1`, the mean equals the shape).

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

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

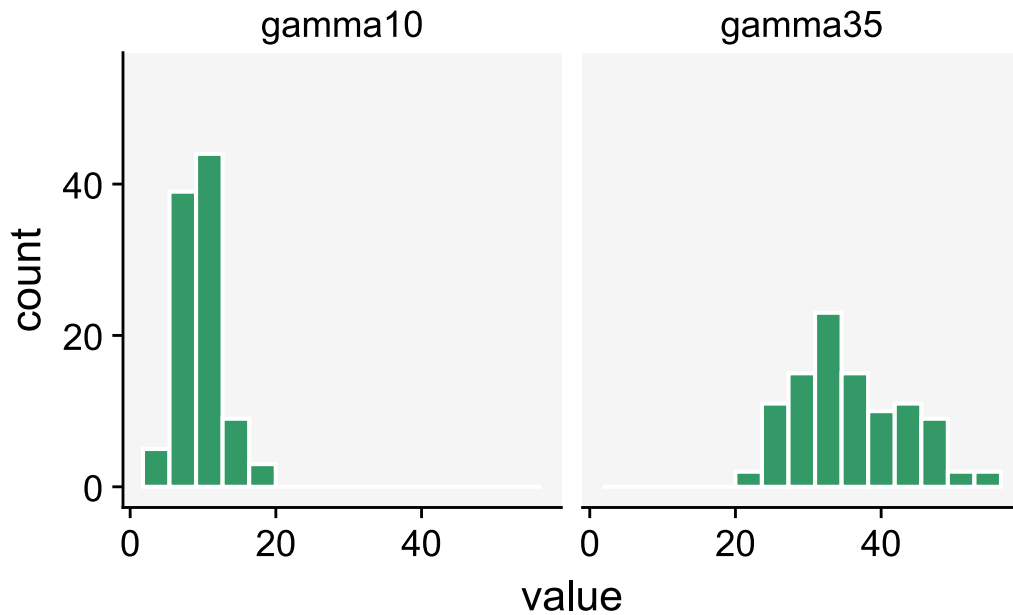


Figure 2.9: Gamma distribution to describe continuous data

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()
```

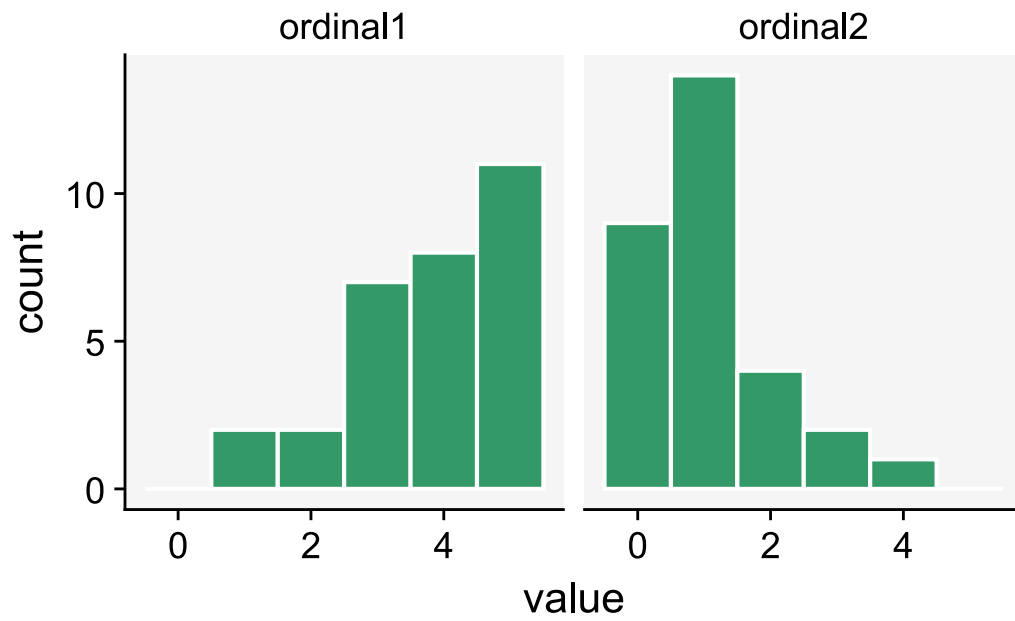


Figure 2.10: Sampling of ordinal data

3. Ordinal scales

Introduction

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., 2007a). In fact, plant pathologists often encounter situations where direct estimates of the nearest percent severity are time-consuming or impractical, besides being estimated unreliably and inaccurately. In plant pathological research, there are two commonly used types of ordinal scales: quantitative and qualitative (Chiang & Bock, 2021).

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., 2007a). 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 & 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 (Chiang et al., 2014; Liu et al., 2019). 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 Horsfall-Barratt 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()

```

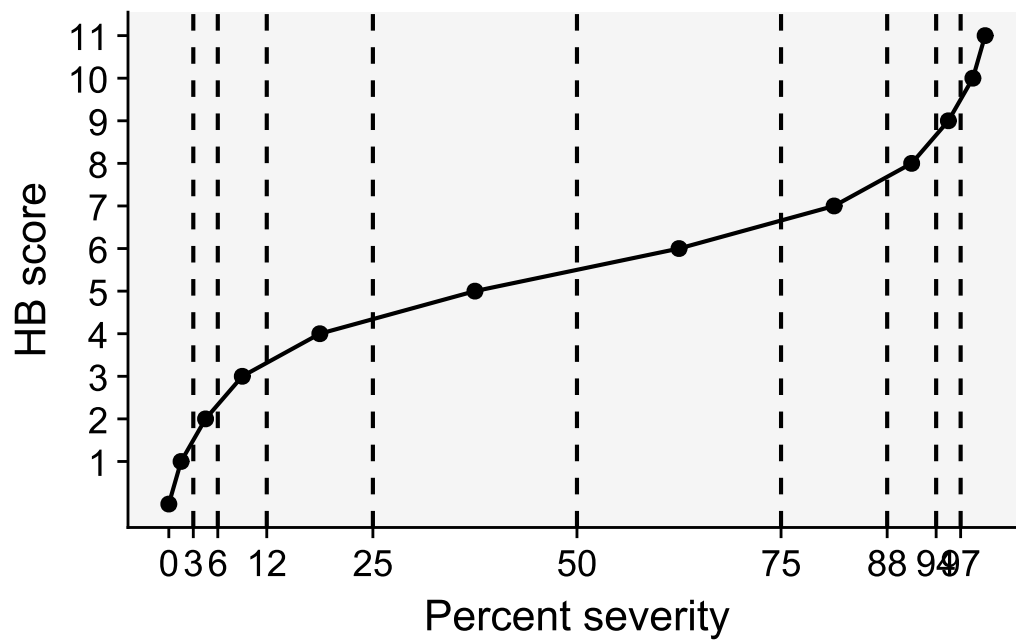


Figure 3.1: Ordinal scores of the Horsfall-Barratt scale

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()

```

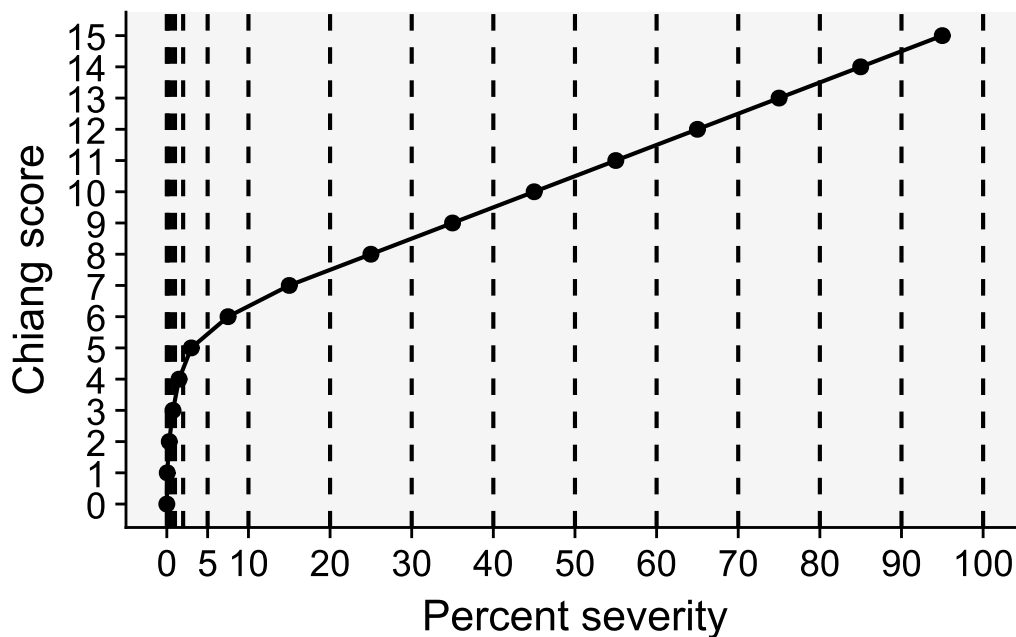


Figure 3.2: Ordinal scores of the Chiang scale

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 & Hollins \(1974\)](#).

Class	Description
0	uninfected
1	slight eyespot (one or more small lesions 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 readily occur)

Disease severity index (DSI)

Sometimes, when quantitative or qualitative ordinal scales are used, the scores given to various individual specimens (the observational units) are transformed into an index on a percentage basis, such as the disease severity index (DSI) which is used as a value for the experimental unit further 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:

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

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

The `DSI()` function automates 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 ratings.

```
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

Analysis of ordinal data

Ordinal score data typically do not align well with the assumptions of traditional parametric statistical methods. Given this challenge, non-parametric methods have emerged as a compelling alternative for handling ordinal plant pathological data (Shah & Madden, 2004). Unlike their parametric counterparts, these methods do not rest on the presumption of a specific distribution for the underlying population, offering greater flexibility in accommodating the intricacies inherent to ordinal data. On the other hand, when the conditions are right, parametric methods can also be harnessed effectively.

A common strategy, particularly for ordinal scores, involves converting these values into the mid-points of their corresponding percent scales. This transformation renders the data more amenable to parametric analyses. However, the mid-point conversion has been criticized in the literature as it may amplify the imprecision, especially when the interval size is wide, and because it does not really reflect a true value, but an interval for each estimate (Chiang et al., 2023; Onofri et al., 2018).

Let's see some examples of analysis using the mid-point conversion in a parametric framework as well as the non-parametric tests.

Example data

We will use a data set made available in an article on the application of the survival analysis technique to test hypotheses when using quantitative ordinal scales (Chiang et al., 2023). The data and codes used in the paper can be found in the specified GitHub repository. The data is structured into four treatments, each with 30 ratings ranging from a score of 1 to 6 on the H-B scale.

```
# Create the vectors for the treatments
trAs <- c(5,4,2,5,5,4,4,2,5,2,2,3,4,3,2,
          2,6,2,2,4,2,4,2,4,5,3,4,2,2,3)
trBs <- c(5,3,2,4,4,5,4,5,4,4,6,4,5,5,5,
          2,6,2,3,5,2,6,4,3,2,5,3,5,4,5)
trCs <- c(2,3,1,4,1,1,4,1,1,3,2,1,4,1,1,
          2,5,2,1,3,1,4,2,2,2,4,2,3,2,2)
trDs <- c(5,5,4,5,5,6,6,4,6,4,3,5,5,6,4,
          6,5,6,5,4,5,5,5,3,5,6,5,5,5,6)

# Create the tibble
dat_ordinal <- tibble::tibble(
  treatment = rep(c("A", "B", "C", "D"),
                  each = 30),
  score = c(trAs, trBs, trCs, trDs)
)

dat_ordinal
```

```
# A tibble: 120 × 2
  treatment score
  <chr>      <dbl>
```

```

1 A      5
2 A      4
3 A      2
4 A      5
5 A      5
6 A      4
7 A      4
8 A      2
9 A      5
10 A     2
# i 110 more rows

```

Because the ordinal response was obtained using an interval scale, the mid-point of the score was also obtained.

```

# Create the vectors for the treatments
trAm <- c(18.5,9,1.5,18.5,18.5,9,9,1.5,
          18.5,1.5,1.5,4.5,9,4.5,1.5,1.5,
          37.5,1.5,1.5,9,1.5,9,1.5,9,18.5,
          4.5,9,1.5,1.5,4.5)
trBm <- c(18.5,4.5,1.5,9,9,18.5,9,18.5,9,
          9,37.5,9,18.5,18.5,18.5,1.5,37.5,
          1.5, 4.5,18.5,1.5,37.5,9,4.5,1.5,
          18.5,4.5,18.5,9,18.5)
trCm <- c(1.5,4.5,0,9,0,0,9,0,0,4.5,1.5,0,9,
          ,0,0,1.5,18.5,1.5,0,4.5,0,9,1.5,1.5,
          1.5,9,1.5,4.5,1.5,1.5)
trDm <- c(18.5,18.5,9,18.5,18.5,37.5,37.5,9,
          37.5,9,4.5,18.5,18.5,37.5,9,37.5,
          18.5,37.5,18.5,9,18.5,18.5,18.5,4.5,18.5,
          37.5,18.5,18.5,18.5,37.5)

# Create the tibble
dat_ordinal_mp <- tibble::tibble(
  treatment = rep(c("A", "B", "C", "D"),
                  each = 30),
  midpoint = c(trAm, trBm, trCm, trDm)
)
dat_ordinal_mp

```

```

# A tibble: 120 × 2
  treatment midpoint
  <chr>         <dbl>
1 A           18.5
2 A            9
3 A           1.5
4 A           18.5
5 A           18.5
6 A            9
7 A            9
8 A           1.5
9 A           18.5
10 A          1.5
# i 110 more rows

```

The visualization using box-plots suggests differences between the treatments.

```
p_score <- dat_ordinal |>
  ggplot(aes(treatment, score))+
  geom_boxplot(outlier.colour = NA)+
  geom_point()+
  theme_r4pde()

p_mp <- dat_ordinal_mp |>
  ggplot(aes(treatment, midpoint))+
  geom_boxplot(outlier.colour = NA)+
  geom_point()+
  theme_r4pde()

library(patchwork)
```

Warning: package 'patchwork' was built under R version 4.5.3

p_score | p_mp

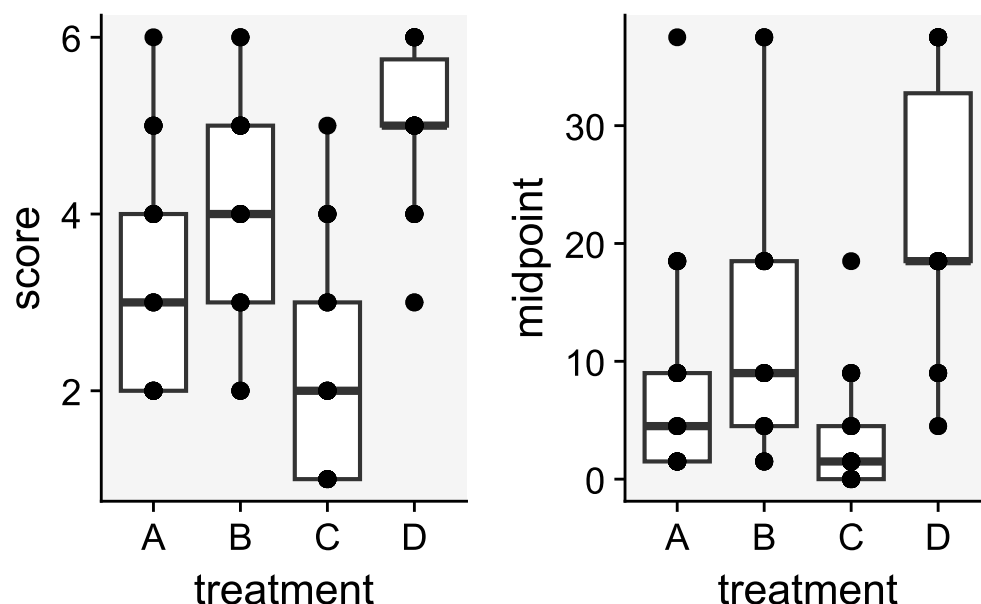


Figure 3.3: Box-plots of the ordinal scores (left) and of the corresponding mid-point percentages (right) for the four treatments

Parametric (mid-point conversion)

Parametric methods can be employed when analyzing data, provided that the underlying assumptions of these methods are satisfied. For data that is based on ordinal scores, one way to apply parametric methods is to convert these scores into the mid-points of their corresponding percent scales. Once converted, the data is better suited for parametric analyses. Among the available techniques, the most frequently utilized is the application of an ANOVA (Analysis of Variance) model. This model is particularly useful for testing the null hypothesis, which posits that there are no significant differences among the treatments being studied.

We can use the `aov` function to fit the model and check the parametric assumptions using the `DHARMa` package.

```
m1_mp <- aov(midpoint ~ treatment, data = dat_ordinal_mp)
# normality and homoscedasticity tests
library(DHARMa)
plot(simulateResiduals(m1_mp))
```

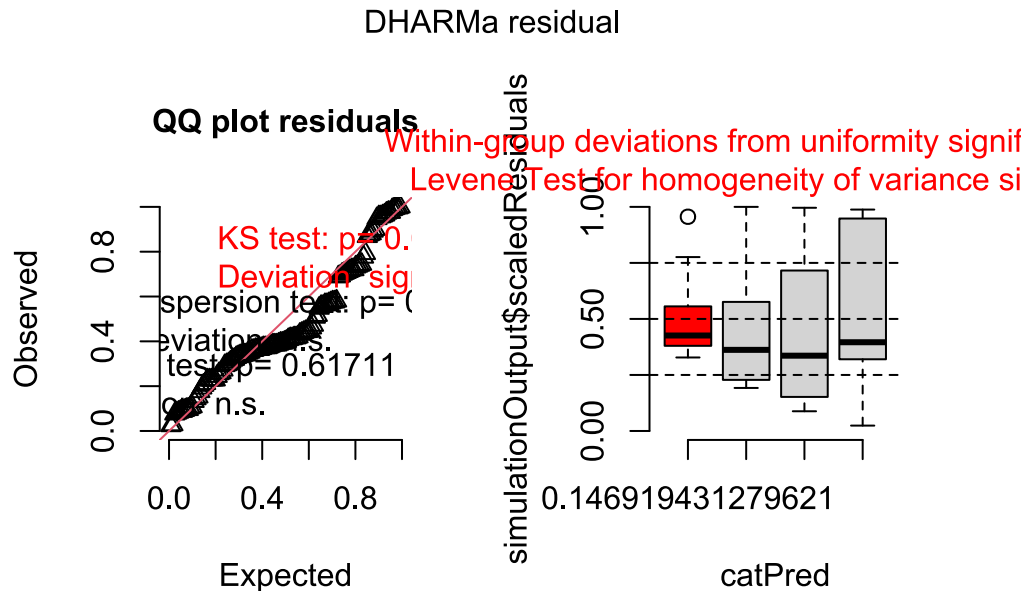


Figure 3.4: Residual diagnostics (DHARMa) of the analysis of variance fitted to the mid-point severity values

Since both normality and homoscedasticity assumptions are violated in our initial model, we will halt further analysis using this model. Instead, we will explore an alternative approach that includes data transformation (logit).

```
# data transformation
library(car)
# transform midpoint to proportion
dat_ordinal_mp$midpoint2 <- dat_ordinal_mp$midpoint/100

m2_mp <- aov(logit(midpoint2) ~ treatment, data = dat_ordinal_mp)
plot(simulateResiduals(m2_mp)) # assumptions are met
```

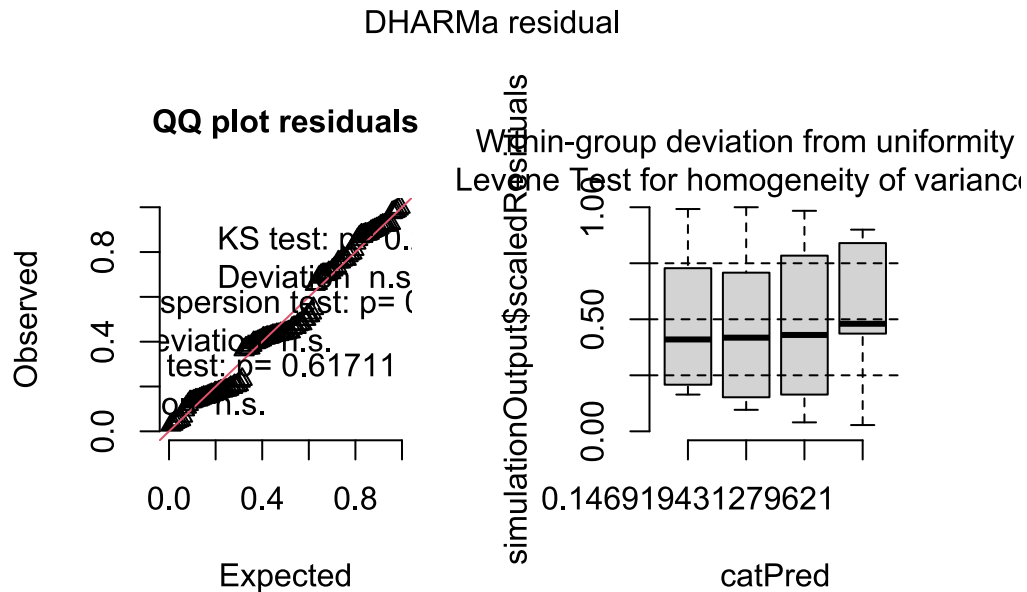


Figure 3.5: Residual diagnostics (DHARMa) of the analysis of variance fitted to the logit-transformed mid-point values

Now that both assumptions have been met, we can proceed to use a means comparison test to determine which treatments differ from one another, with the assistance of the `{emmeans}` package. It's important to note that the results are displayed in the original scale after transformation, specifically when using `type = "response"`.

```
library(emmeans)
```

```
Welcome to emmeans.
```

```
Caution: You lose important information if you filter this package's results.
```

```
See '? untidy'
```

```
means_m2_mp <- emmeans(m2_mp, "treatment", type = "response")
means_m2_mp
```

treatment	response	SE	df	lower.CL	upper.CL
A	0.0808	0.00983	116	0.0634	0.1026
B	0.1248	0.01450	116	0.0989	0.1564
C	0.0461	0.00582	116	0.0358	0.0591
D	0.2071	0.02170	116	0.1674	0.2535

```
Confidence level used: 0.95
```

```
Intervals are back-transformed from the logit scale
```

The best way to visualize the differences between treatments is using the `pwpm` function to display a matrix of estimates, pairwise differences, and P values. Most commonly, the compact letter display is used for means comparison.

```
# matrix
pwpm(means_m2_mp)
```

```

      A      B      C      D
A [0.0808] 0.0530 0.0094 <.0001
B 0.617 [0.1248] <.0001 0.0085
C 1.820 2.953 [0.0461] <.0001
D 0.337 0.546 0.185 [0.2071]

Row and column labels: treatment
Upper triangle: P values null = 1 adjust = "tukey"
Diagonal: [Estimates] (response) type = "response"
Lower triangle: Comparisons (odds.ratio) earlier vs. later

```

```

# compact letter display
library(multcomp)
cld(means_m2_mp)

```

```

treatment response      SE df lower.CL upper.CL .group
C           0.0461 0.00582 116  0.0358  0.0591  1
A           0.0808 0.00983 116  0.0634  0.1026  2
B           0.1248 0.01450 116  0.0989  0.1564  2
D           0.2071 0.02170 116  0.1674  0.2535  3

Confidence level used: 0.95
Intervals are back-transformed from the logit scale
P value adjustment: tukey method for comparing a family of 4 estimates
Tests are performed on the log odds ratio scale
significance level used: alpha = 0.05
NOTE: If two or more means share the same grouping symbol,
      then we cannot show them to be different.
      But we also did not show them to be the same.

```

Non-parametric (ordinal score)

Because ordinal score data generally do not meet the assumptions of traditional parametric statistical methods, non-parametric methods can be considered as an alternative. Such methods have been proposed for analyzing ordinal data in plant pathology (Shah & Madden, 2004). For our example, when more than two treatments are involved, a Kruskal-Wallis test can be utilized. The `kruskal` function of the `{agricolae}` package does the job we want. Note that in this case we use the ordinal score directly and not the mid-point value.

```

library(agricolae)
kruskal(dat_ordinal$score, dat_ordinal$treatment, console = TRUE)

```

```

Study: dat_ordinal$score ~ dat_ordinal$treatment
Kruskal-Wallis test's
Ties or no Ties

Critical Value: 52.34422
Degrees of freedom: 3
Pvalue Chisq : 2.529543e-11

dat_ordinal$treatment, means of the ranks

dat_ordinal.score r

```



```

A      52.05000 30
B      69.58333 30
C      29.61667 30
D      90.75000 30

Post Hoc Analysis

t-Student: 1.980626
Alpha    : 0.05
Minimum Significant Difference: 13.1991

Treatments with the same letter are not significantly different.

dat_ordinal$score groups
D      90.75000      a
B      69.58333      b
A      52.05000      c
C      29.61667      d

```

Survival analysis

A method based on survival analysis was more recently introduced for the analysis of ordinal data ([Chiang et al., 2023](#)). The authors developed an R script called "CompMuCens", which facilitates the comparison of multiple treatment means for plant disease severity data. This is achieved through nonparametric survival analysis using class- or interval-based data derived from a quantitative ordinal scale. The code is accessible in [this repository](#) and has been incorporated into the {r4pde} package.

We continue with our working example data. Since the expected variable name for the score in the function is `x`, we must adjust our data frame accordingly.

```
names(dat_ordinal) <- c("treatment", "x")
```

The `CompMuCens()` function uses the `icctest()` from the {interval} package to conduct nonparametric survival analysis. Detailed explanation of the function's input and output can be found [here](#). We just need to set the `dat` and the `scale` arguments. The scale will be used to convert the scores into the defined interval which is used as response variable in the analysis. For example, if the score is 2, the respective limits of the interval will be 3 and 6. For 7, the limits will be 75 and 88. The function takes care of this conversion based on the inputted scale values.

```
library(interval)
```

```
Loading required package: perm
```

```
Loading required package: Icens
```

```
Loading required package: MLEcens
```

```
Depends on Icens package available on bioconductor.
To install use for example:
```

```
install.packages('BiocManager')
BiocManager::install('Icens')
```

```
library(r4pde)
```

```
scale <- c(0,3,6,12,25,50,75,88,94,97,100, 100)
CompMuCens(dat_ordinal, scale)
```

```
$U.Score
# A tibble: 4 × 2
  treatment score
  <chr>      <dbl>
1 D        -15.1
2 B         -4.54
3 A          4.23
4 C         15.4

$Hypothesis.test
# A tibble: 3 × 4
  treat1 treat2 `p-value for H0: treat1 <= treat2` p-value for H0: treat1 = tr...1
  <chr>  <chr>                                <dbl> <lgl>
1 D      B                                0.00188 NA
2 B      A                                0.0113  NA
3 A      C                                0.000746 NA
# i abbreviated name: 1`p-value for H0: treat1 = treat2`

$adj.Signif
[1] 0.01666667

$Conclusion
[1] "c(\"D>B\", \">A\", \">C\")"
```

There are three outputs: the ordered scores for each treatment, the pairwise comparison between treatments and the significance levels, followed by a conclusion section. In this example all treatments differ from each other.

Interpretation

Upon comparing the three methods, it becomes evident that there is a marked distinction in their outcomes. The Kruskal-Wallis and the survival analysis tests suggested a significant difference between treatment A and B, whereas the parametric counterpart did not. However, it's crucial to note that the P-value is borderline significant at 0.0530, being just slightly above the conventional threshold of 0.05. Given this, it appears that the conversion to the mid-point might have resulted in a type II error, wherein a genuine difference between the treatments might have been missed or overlooked. A detailed comparison between the mid-point and survival analysis has been presented in the paper, which is worth reading ([Chiang et al., 2023](#)).

4. Measuring foliar severity

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 (see the chapter on [reliability and accuracy](#)), 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.

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, we 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, we 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.

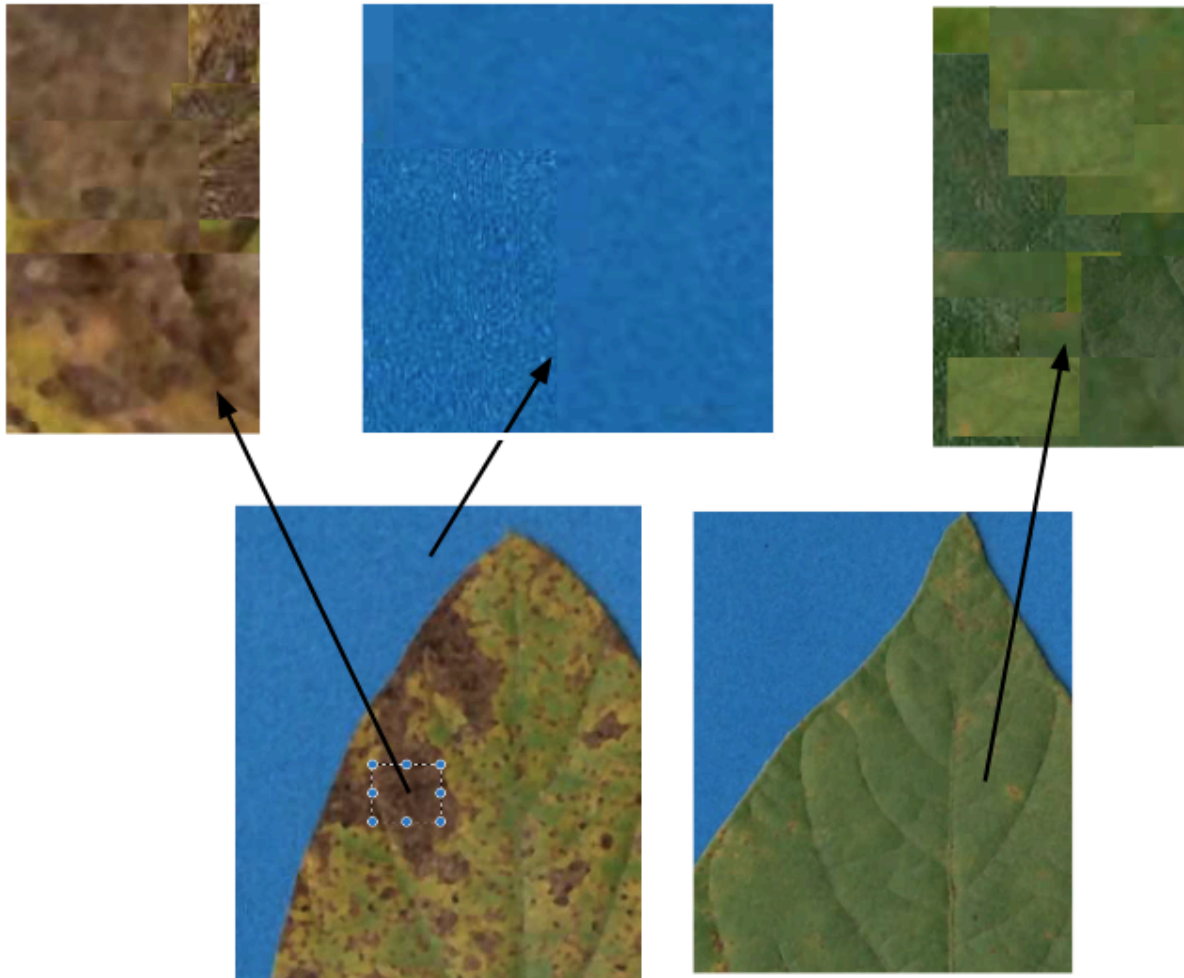


Figure 4.1: Preparation of image palettes by manually sampling fractions of the images that represent background, healthy leaf and lesions

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)
```

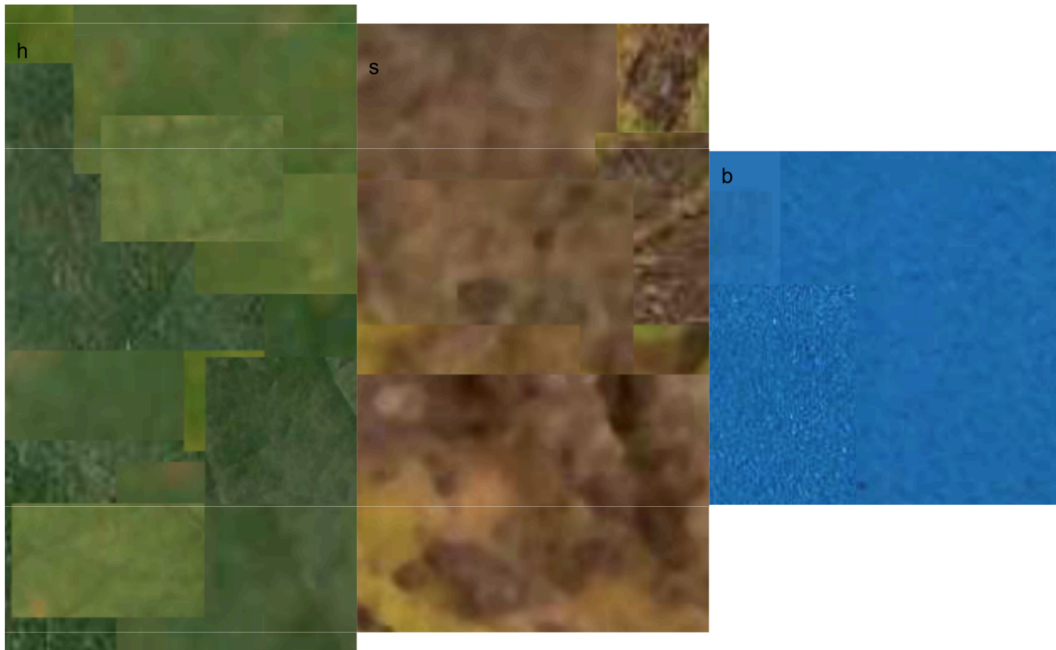


Figure 4.2: Image palettes created to segment images into background, symptoms and healthy area of the image

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

```
img <- image_import("imgs/originals/img46.png")  
  
b <- pick_palette(img, viewer = "mapview")
```

The original image is displayed and the user needs to select the place marker. It is possible to zoom in the image. After placing the markers via clicking on the background colors multiple times, the user should click on "Done".



Figure 4.3: Soybean leaf displaying symptoms of soybean rust

```
image_combine(b)
```



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

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

```
# Symptoms
s <- pick_palette(img, viewer = "mapview")

# healthy
h <- pick_palette(img, viewer = "mapview")
```

Measuring severity

Single image

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)
```

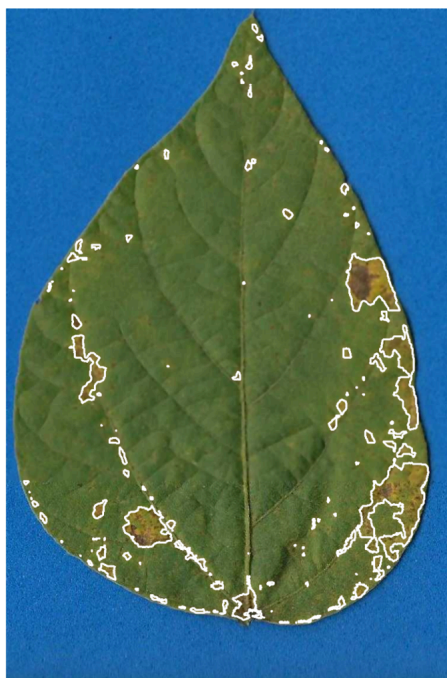


Figure 4.5: Imported image for further analysis

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
)
```

```
i Processing a single image. Please, wait.
```

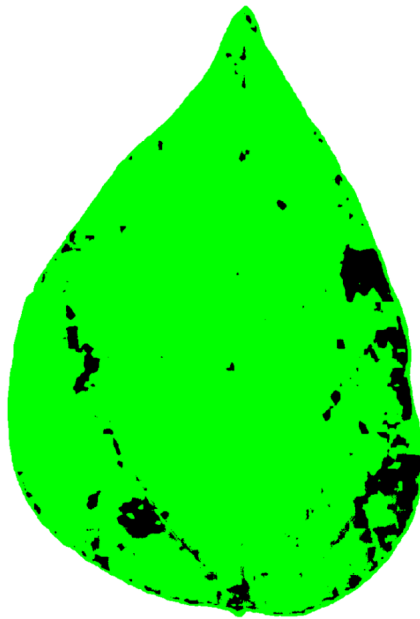



✓ Image Successfully analyzed! [1.5s]

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

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

i Processing a single image. Please, wait.



✓ Image Successfully analyzed! [2.2s]

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
)
```

— Processing successfully finished — on 2026-10-04 | 20:22:17 —

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

```
  img healthy symptomatic
1 img11 70.78468 29.2153200
2 img35 46.91117 53.0888304
3 img37 60.45702 39.5429797
4 img38 79.13170 20.8683005
5 img46 93.14952  6.8504782
6 img5  20.49773 79.5022701
7 img63 97.15526  2.8447389
8 img67 99.83711  0.1628887
9 img70 35.52691 64.4730939
10 img75 93.04095  6.9590466
```

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

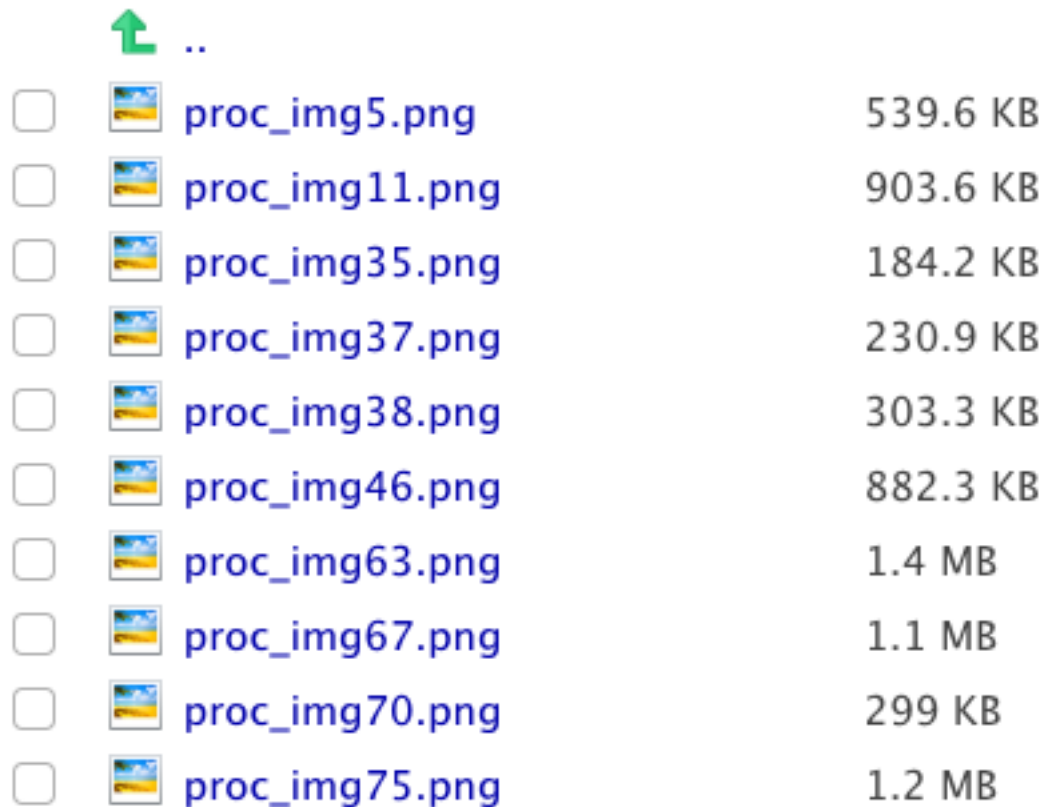


Figure 4.6: Images created by pliman and exported to a specific folder

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

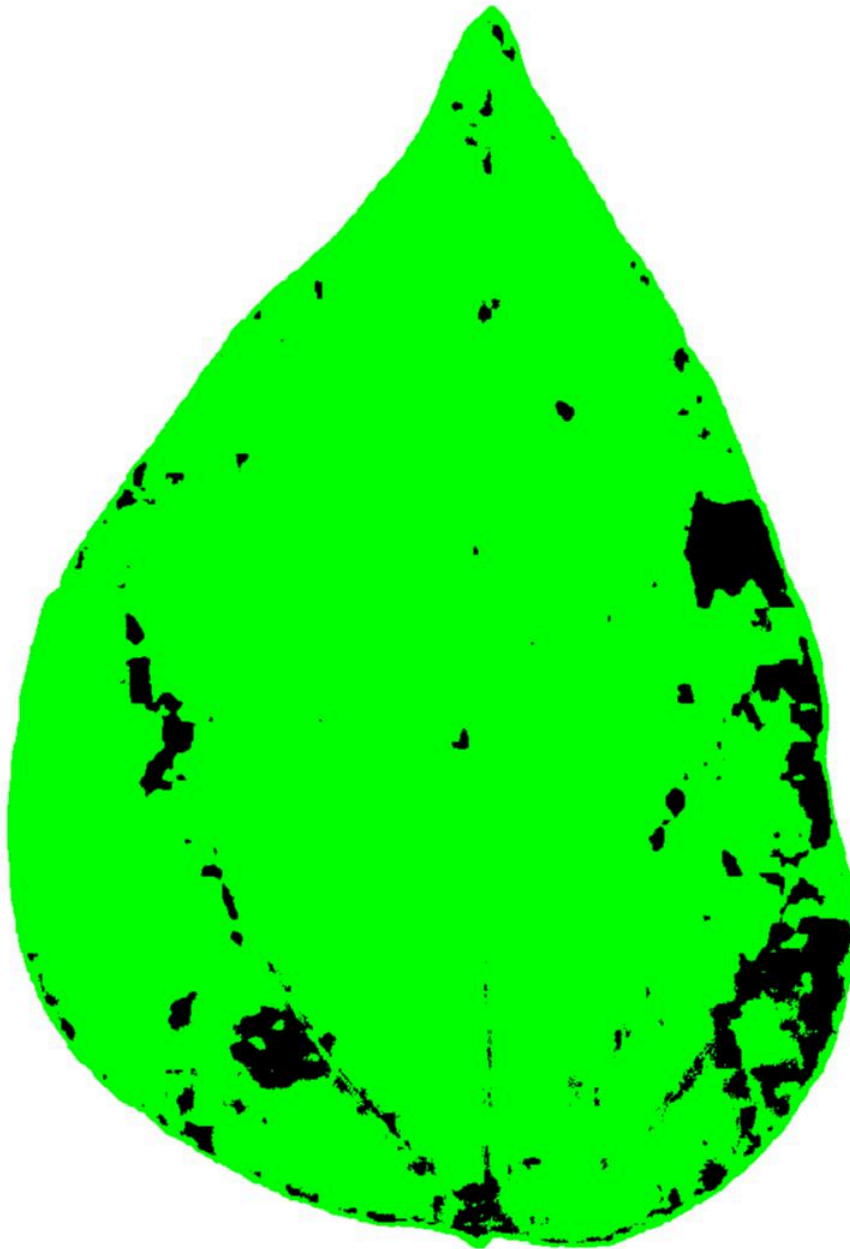


Figure 4.7: Figure created by `pliman` after batch processing to segment the images and calculate percent area covered by symptoms. The symptomatic area is delineated in the image.

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 specify the directory for the processed images using `dir_processed`.

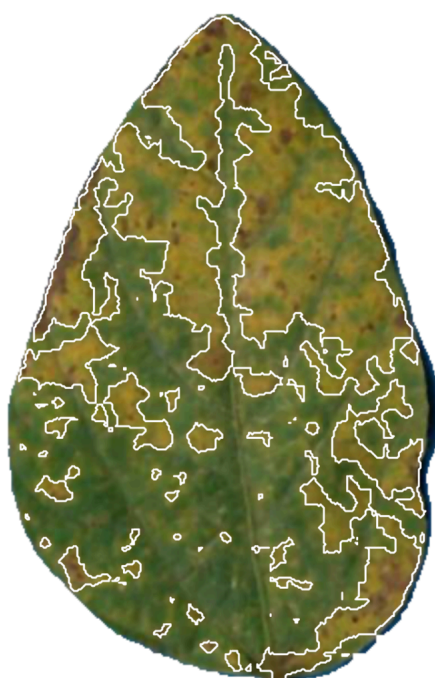
```
img2 <- image_import("imgs/soybean_three.png")
image_combine(img2)
```



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")
```

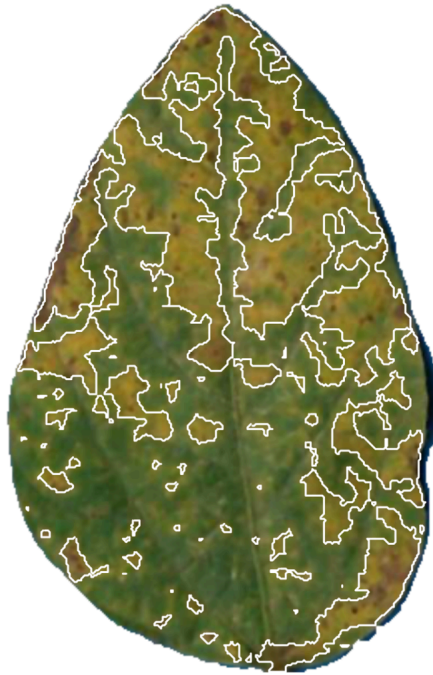
```
i Processing a single image. Please, wait.  
i Processing a single image. Please, wait.
```



✓ Image Successfully analyzed! [597ms]

i Processing a single image. Please, wait.

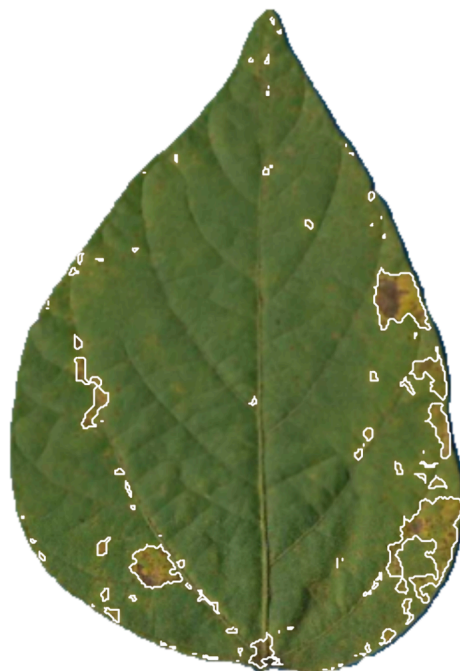
i Processing a single image. Please, wait.



✓ Image Successfully analyzed! [698ms]

i Processing a single image. Please, wait.

i Processing a single image. Please, wait.



```
✓ Image Successfully analyzed! [919ms]

i Processing a single image. Please, wait.
✓ Image Successfully analyzed! [2.6s]
```

```
pliman2$severity
```

```
img leaf healthy symptomatic
1 img 1 59.20586 40.794142
2 img 2 61.16515 38.834847
3 img 3 93.21289 6.787109
```

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



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

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,
  "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()
```

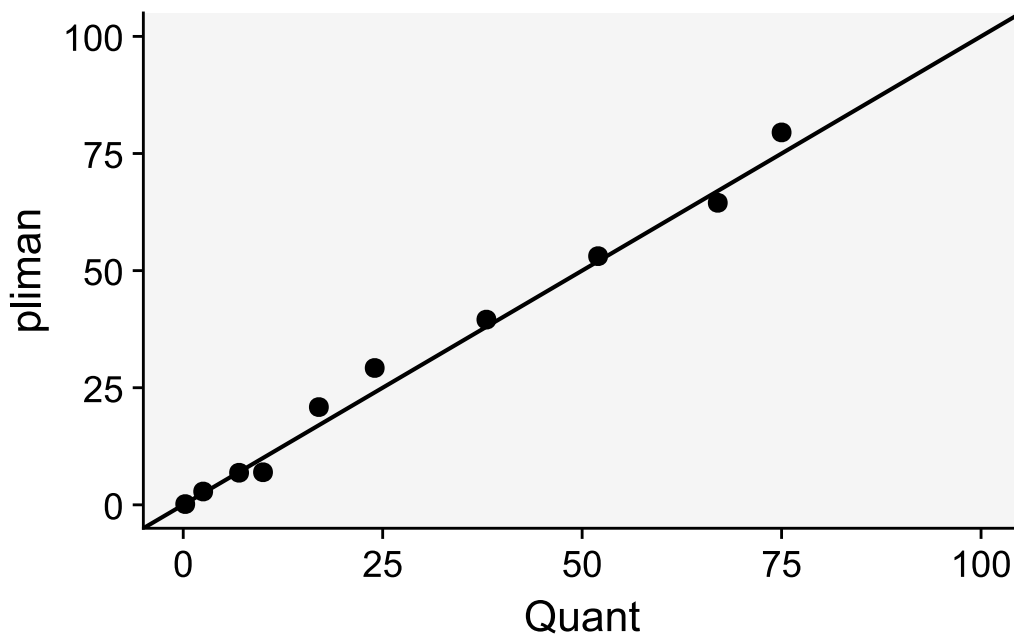


Figure 4.10: Scatter plot for the relationship between severity values measured by pliman and Quant software

The concordance correlation coefficient is an index of agreement between two observers or two methods (see the chapter on [reliability and accuracy](#)). 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 0.9940583 0.9774009 0.9984474
```


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 other image-analysis tools, namely, the need for images to be taken under uniform and controlled conditions, especially with a contrasting background.

5. Reliability and accuracy

Severity data

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 occurs. 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).

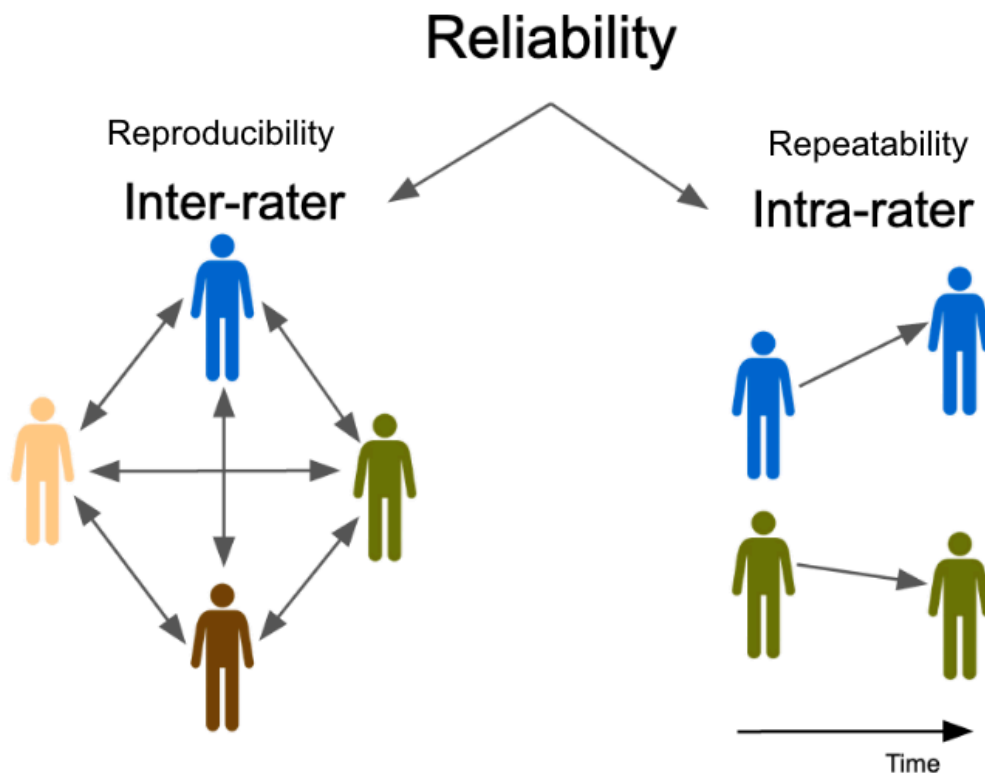


Figure 5.1: Two types of reliability of estimates or measures of plant disease intensity

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.

```
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)
```

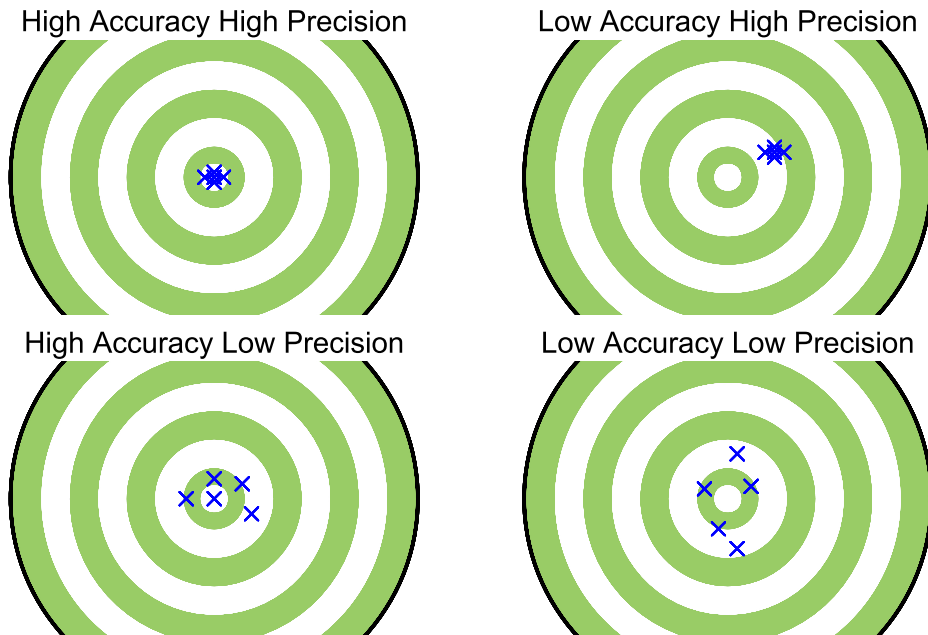


Figure 5.2: The accuracy and precision of the archer is determined by the location of the group of arrows

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

```
library(tidyverse)
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, linewidth = 1)+
  geom_smooth(method = "lm")+
  geom_point(color = "#99cc66", size = 3)+
  ylim(0,70)+
  xlim(0,70)+
  labs(x = "Actual", y = "Estimate",
       subtitle = "High Accuracy High Precision")

ip <- dat |>
  ggplot(aes(actual, ip))+
  geom_abline(intercept = 0, slope = 1,
              linetype = 2, linewidth = 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 Accuracy High Precision")

ai <- dat |>
  ggplot(aes(actual, ai))+
  geom_abline(intercept = 0, slope = 1,
              linetype = 2, linewidth = 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 Accuracy Low Precision")

ii <- dat |>
  ggplot(aes(actual, ii))+
  geom_abline(intercept = 0, slope = 1,
              linetype = 2, linewidth = 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 Accuracy Low Precision")

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

```

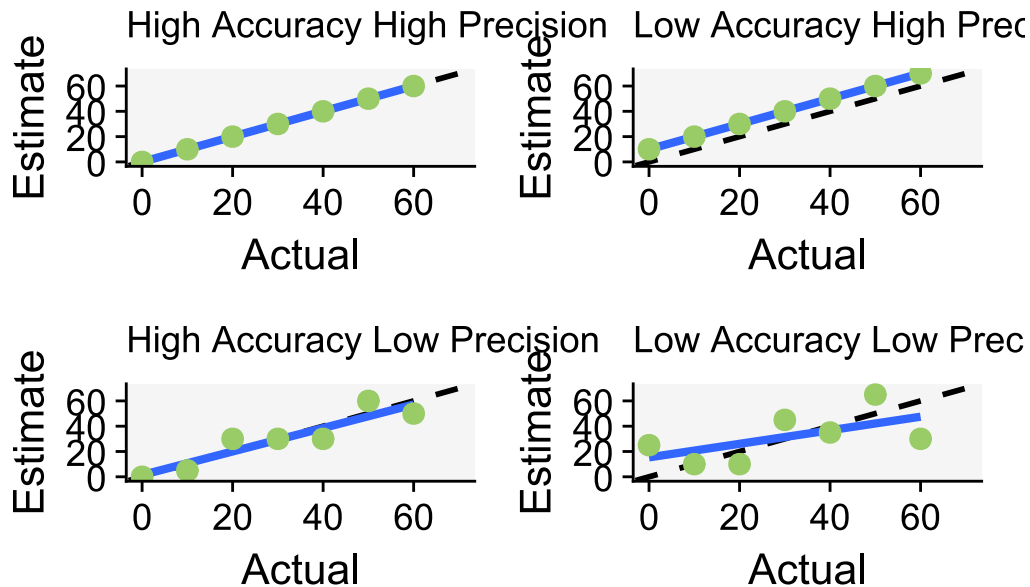


Figure 5.3: Scatter plots for the relationship between actual and estimated values representing situations of low or high precision and accuracy. The dashed line indicates the perfect concordance and the solid blue line represents the fit of the linear regression model

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 hypotheses, such as whether one method is superior to the other. The indices or the tests vary according to the nature of the variable, whether continuous, binary or categorical.

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 (Franceschi et al., 2020). 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 corresponds to one symptomatic leaf. Let's assign the tibble to a dataframe called `sbr` (an acronym for soybean rust). Note that the variable is continuous.

```
library(tidyverse)
sbr <- tribble(
  ~leaf, ~R1, ~R2, ~R3, ~R4,
  1, 0.6, 0.6, 0.7, 0.6,
  2, 2, 0.7, 5, 1,
  3, 5, 5, 8, 5,
  4, 2, 4, 6, 2,
  5, 6, 14, 10, 7,
  6, 5, 6, 10, 5,
  7, 10, 18, 12.5, 12,
  8, 15, 30, 22, 10,
```

```

9, 7, 2, 12, 8,
10, 6, 9, 11.5, 8,
11, 7, 7, 20, 9,
12, 6, 23, 22, 14,
13, 10, 35, 18.5, 20,
14, 19, 10, 9, 10,
15, 15, 20, 19, 20,
16, 17, 30, 18, 13,
17, 19, 53, 33, 38,
18, 17, 6.8, 15, 9,
19, 15, 20, 18, 16,
20, 18, 22, 24, 15
)

```

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

```

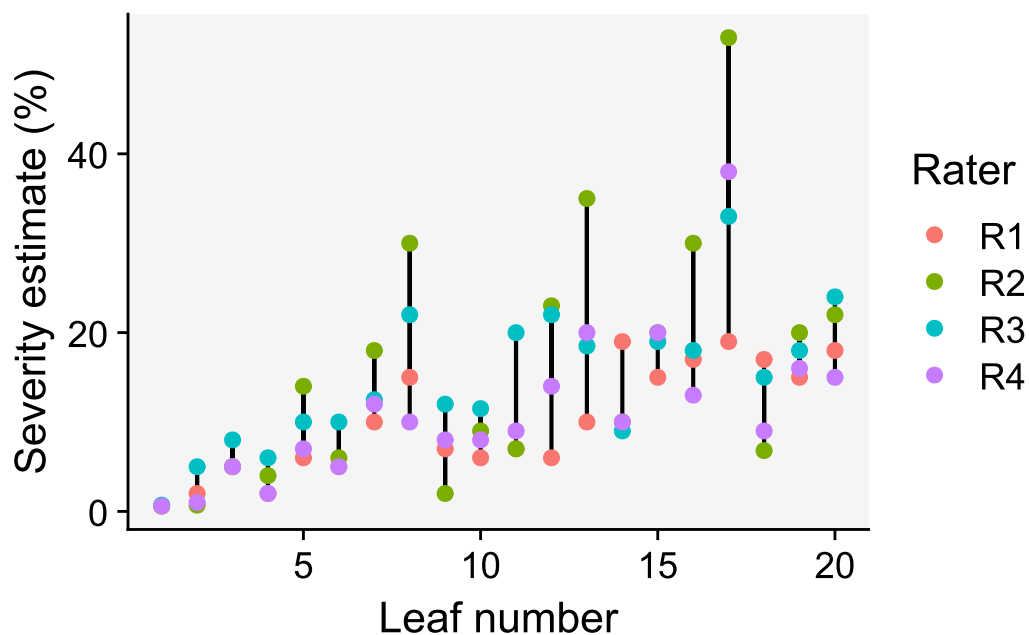


Figure 5.4: Visual estimates of soybean rust severity for each leaf by each of four raters

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()
```

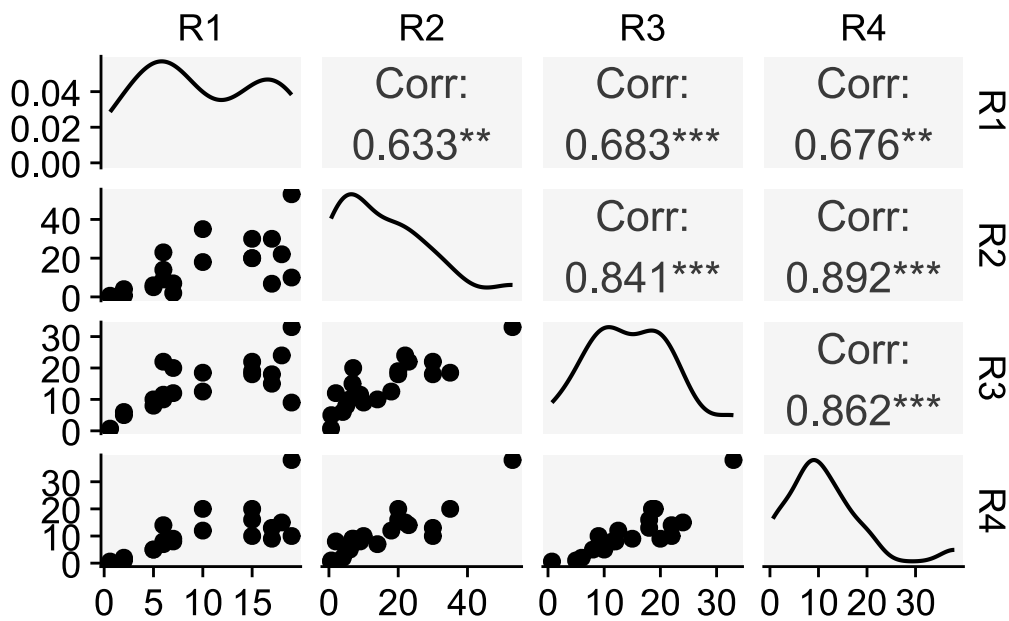


Figure 5.5: Correlation plots relating severity estimates for all pairs of raters

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,

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

Table 5.1: 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)

# 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

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

[1] 0.595
```

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 depends on the particular experimental design. Following the convention of the seminal work by [Shrout & 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()` function of the *psych* package. It provides all six forms of the ICC, with the F value and the lower and upper bounds of the 95% confidence interval.

```
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
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 data frame with the results of all distinct ICCs (we showed only the first two columns above). The first three rows are the single-rater forms: ICC1 is the one-way model, ICC2 is the two-way random effects model (absolute agreement) and ICC3 is the two-way mixed model (consistency). The last three rows (ICC1k, ICC2k and ICC3k) are the corresponding forms for the average of the k raters. We note that ICC1 and ICC2 gave very close results. The full output, including the confidence intervals, can be shown with:

```
knitr::kable(ic$results[, c("type", "ICC", "lower bound", "upper bound")], digits = 3)
```

	type	ICC	lower bound	upper bound
Single_raters_absolute	ICC1	0.641	0.440	0.813
Single_random_raters	ICC2	0.646	0.435	0.819
Single_fixed_raters	ICC3	0.692	0.503	0.845
Average_raters_absolute	ICC1k	0.877	0.758	0.946
Average_random_raters	ICC2k	0.880	0.755	0.948
Average_fixed_raters	ICC3k	0.900	0.802	0.956

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.occ` function of the `epiR` package does the job but it does not compute a confidence interval.

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

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

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.

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 5.6). To facilitate, we need the data in the long format.

```
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 = "Estimated severity (%)")
```

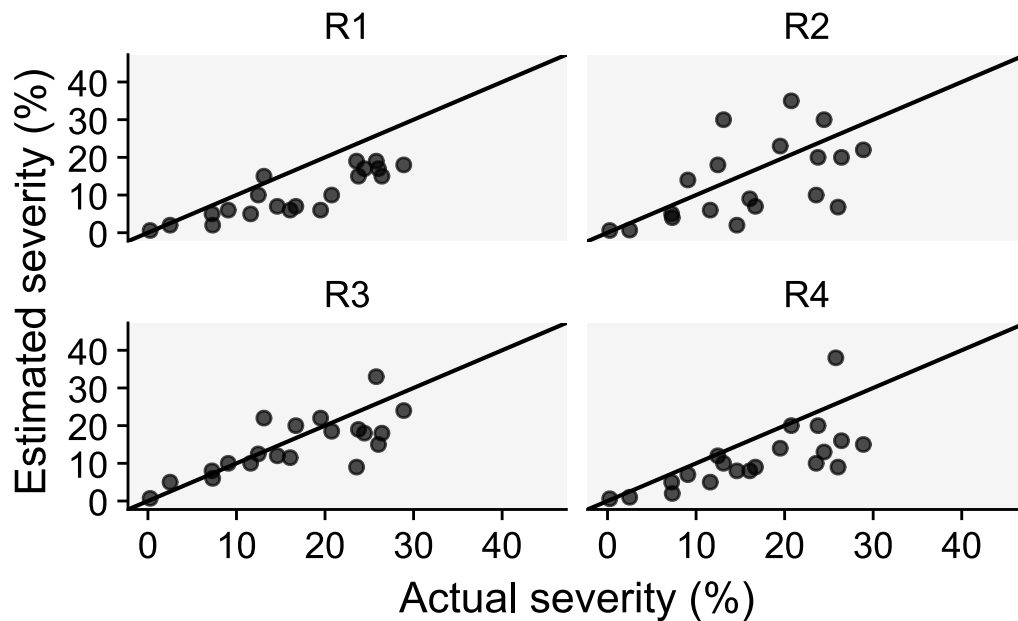


Figure 5.6: Scatterplots for the relationship between estimated and actual severity for each rater

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
```

Accuracy

Absolute errors

It is useful to visualize the errors of the estimates which are obtained by subtracting the actual severity values from the estimates. 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)")
```

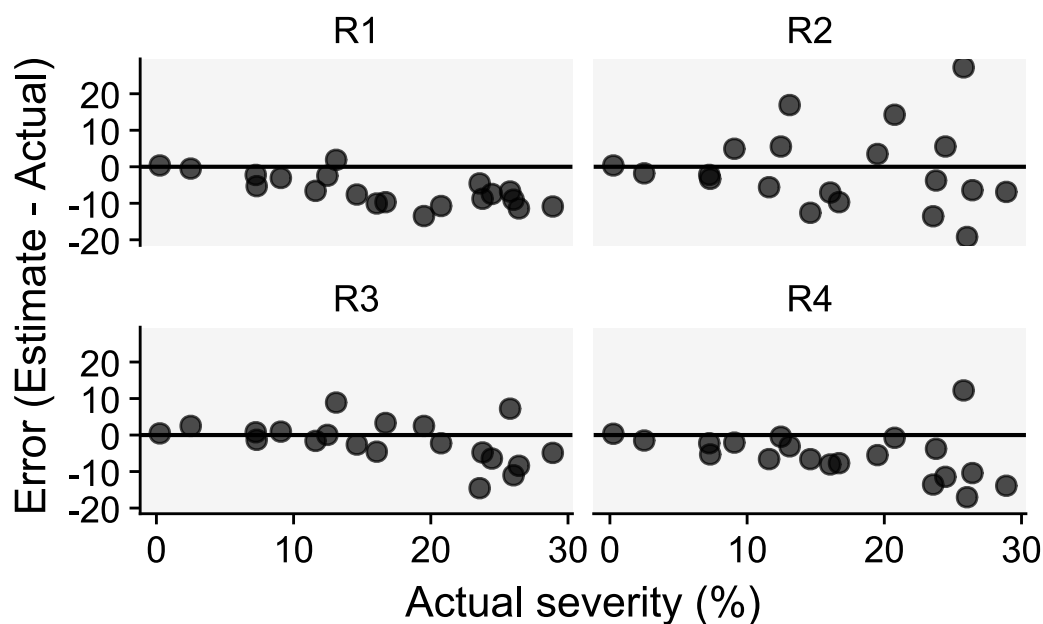


Figure 5.7: Error (estimated - actual) of visual severity estimates

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$rho.c[,1]/ccc$C.b
# 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 5.2: Statistics 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	0.5845291
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	0.8725643	1.3652694	0.9090386
R2	0.5230656	0.8948494	0.5845291	0.6214585	0.0666069
R3	0.7306948	0.9701226	0.7531983	1.1028813	0.2280303
R4	0.5861371	0.8245860	0.7108260	1.0044929	0.6522573

Incidence data

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 classifications 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 negative.

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 what a confusion matrix looks like.

Suppose a 2×2 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)
cm <- confusionMatrix(factor(dat_inc$class), reference = factor(dat_inc$actual),
                        positive = "1")
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 : 0.8571
      Neg Pred Value : 0.6000
      Prevalence : 0.6667
      Detection Rate : 0.5000
      Detection Prevalence : 0.5833
      Balanced Accuracy : 0.7500

      'Positive' Class : 1
```

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 = 6
FP = 1
FN = 2
TN = 3
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(dat_inc$actual, dat_inc$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
```

Reliability

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

```
[1] 0.48
```

6. Standard area diagrams

Definitions

According to a glossary on phytopathometry (Bock et al., 2021a), 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:

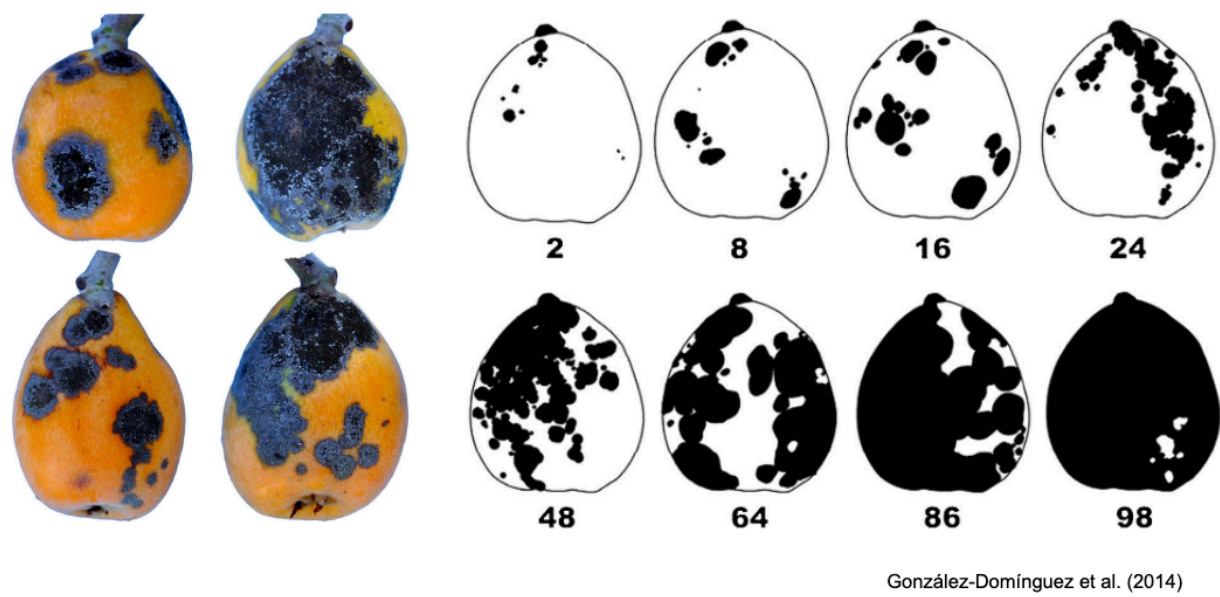


Figure 6.1: Actual photos of symptoms of loquat scab on fruit (left) and a SAD with eight diagrams (right). Each number represents severity as the percent area affected (González-Domínguez et al., 2014)

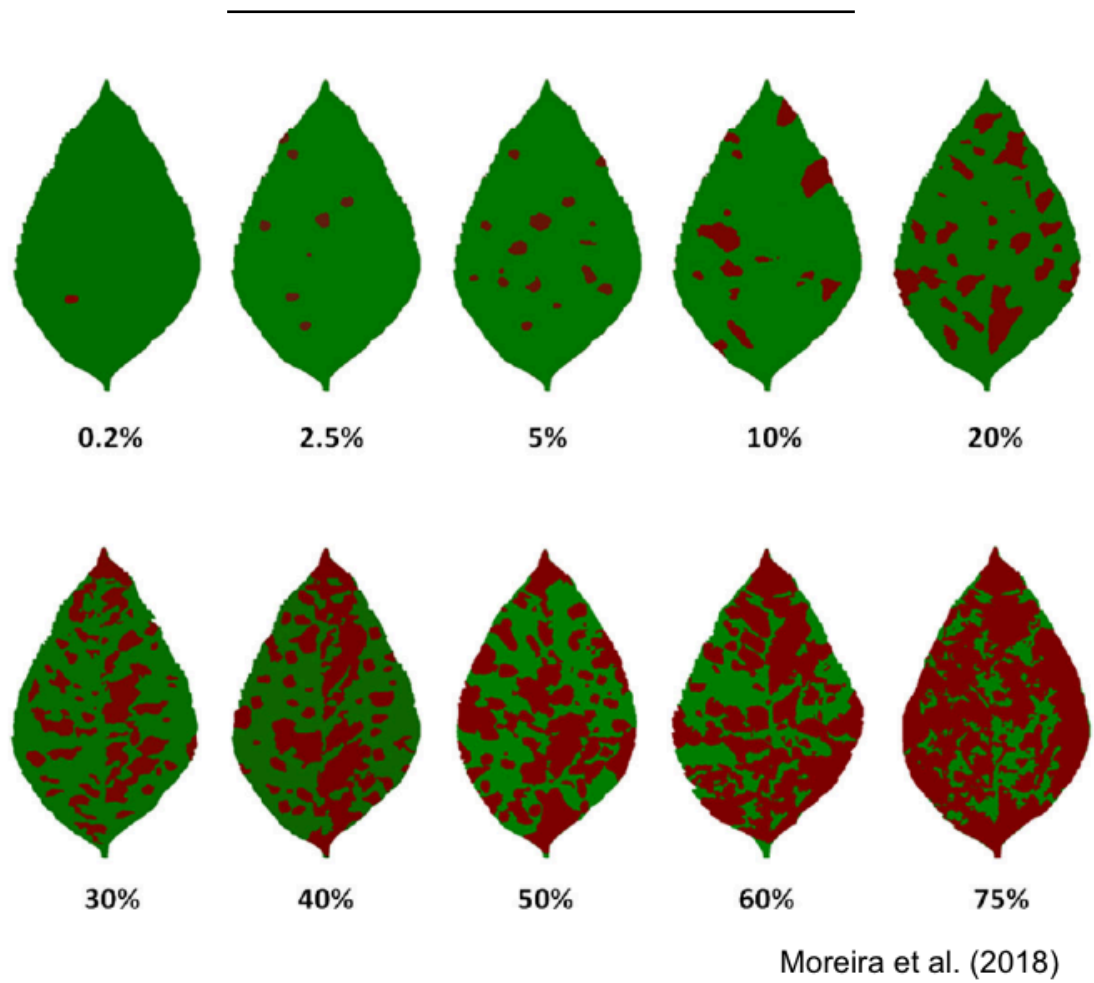
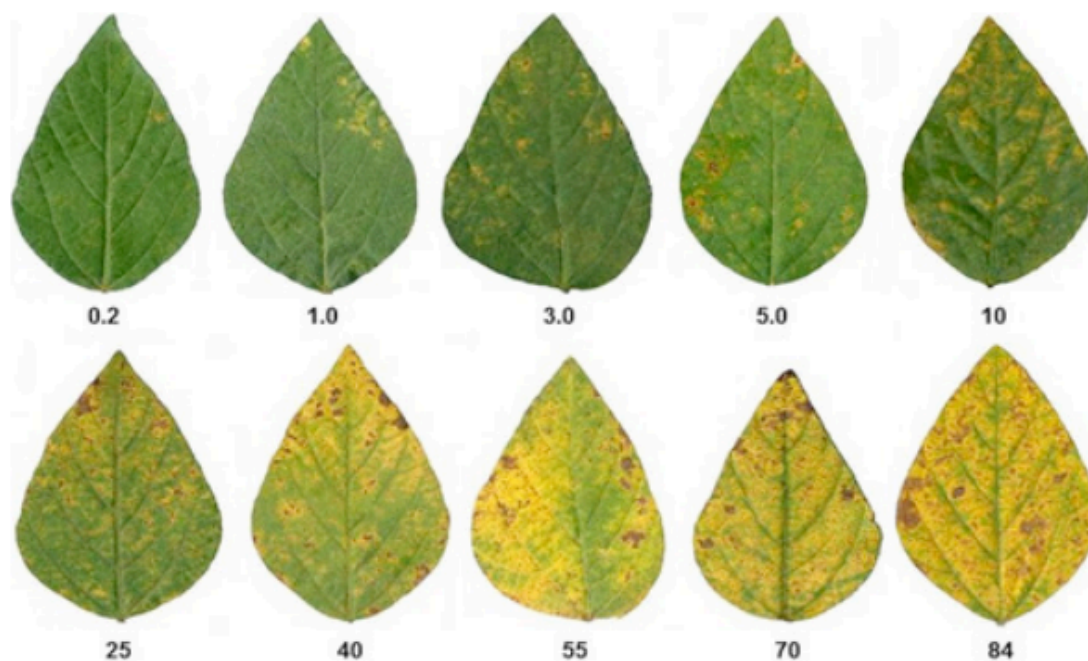


Figure 6.2: SADs for Glomerella leaf spot on apple leaf. Each number represents severity as the percent area affected (Moreira et al., 2018)



Franceschini et al. (2020)

Figure 6.3: SADs for soybean rust. Each number represents severity as the percent area affected (Franceschi et al., 2020)

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.

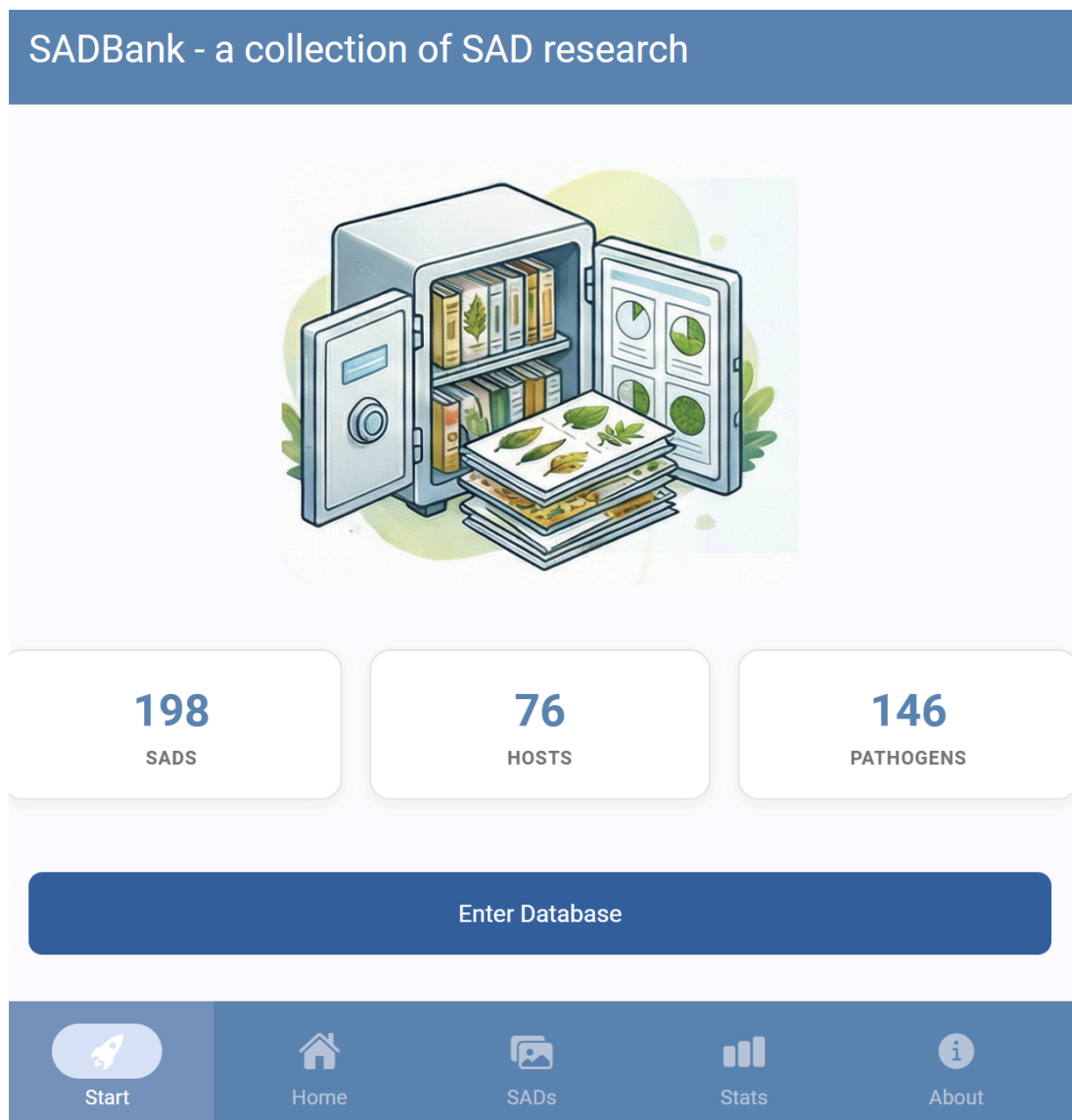


Figure 6.4: SADBank, a curated collection of articles

SAD development and validation

A systematic review of the literature on SADs highlighted the most important aspects related to 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. The most important aspects to note are:

💡 Best practices for SAD 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.
- 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 post-test experiment conditions should be the same to avoid any impact of distraction on accuracy of estimates during the tests.

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 SAD with a pre-defined number of diagrams. Firstly, as shown in the [chapter on measuring foliar severity](#), the set of images to be selected needs to be analyzed using the `measure_disease()` function. Then, a SAD 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 that 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")
```

```

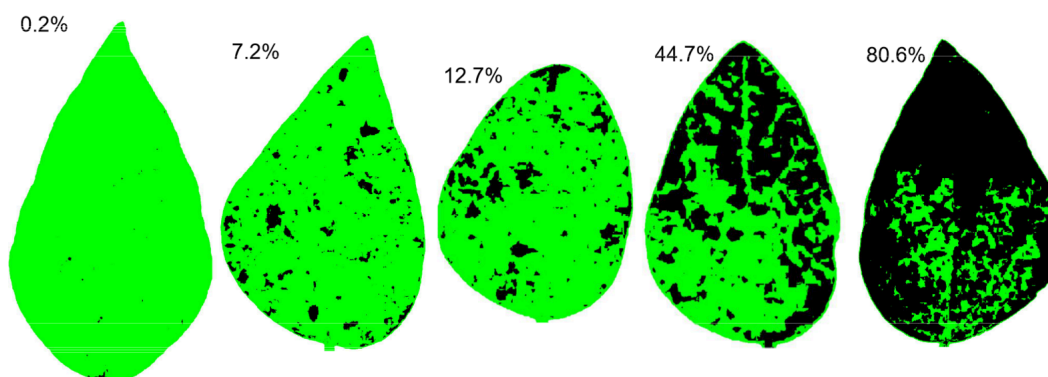
b <- image_import("imgs/sbr_b.png")

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 SAD with five diagrams side by side. The resulting SAD is in two colors by default. Set the argument `show_original` to `TRUE` for showing the original image in the SAD.

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



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 analyze SADs validation data where raters are taken as a random effect in the model (Franceschi et al., 2020; González-Domínguez et al., 2014; Pereira et al., 2020).

Nonparametric bootstrapping 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 H_0 , 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)
```

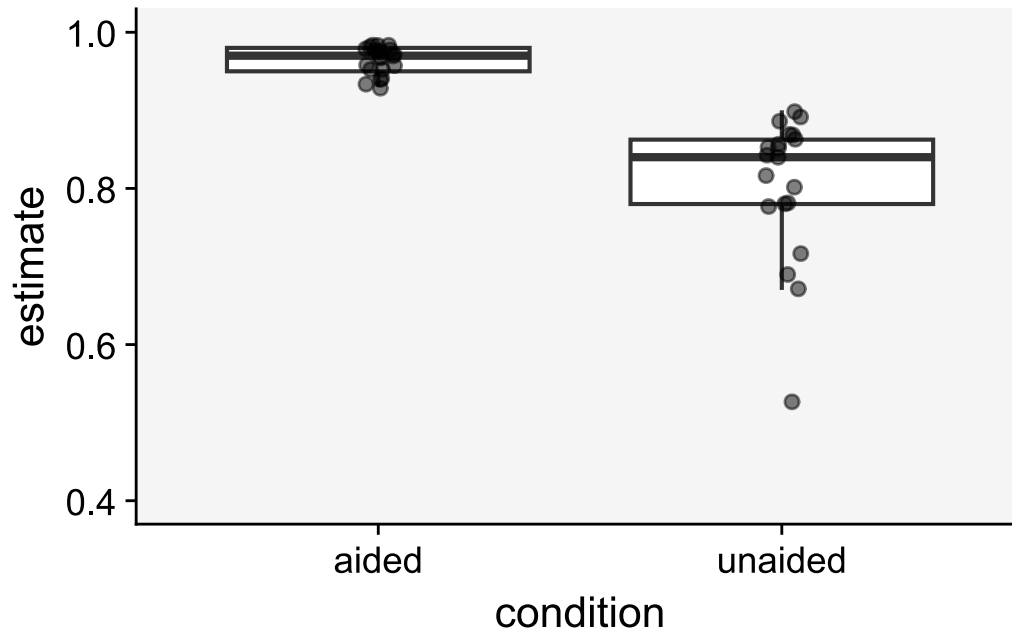


Figure 6.5: Concordance correlation coefficient (CCC) of 20 raters estimating severity without (unaided) and with (aided) the standard area diagram. Each point is a rater

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")
```

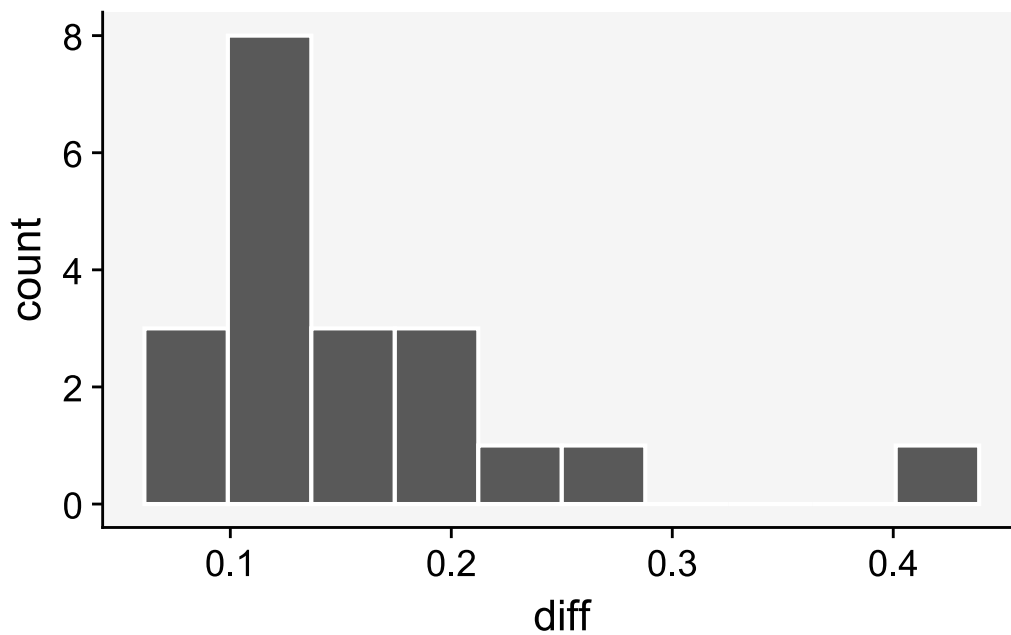


Figure 6.6: Histogram of the differences in CCC between the aided and unaided conditions (aided minus unaided)

Using the simpleboot and boot packages of R:

```
library(simpleboot)
```

Warning: package 'simpleboot' was built under R version 4.5.3

Simple Bootstrap Routines (1.1-8)

```
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%	(0.1254, 0.1937)	(0.1220, 0.1905)

Level	Percentile	BCa
95%	(0.1285, 0.1970)	(0.1285, 0.1970)

Calculations and Intervals on Original Scale

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

```
[1] 0.1595
```

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

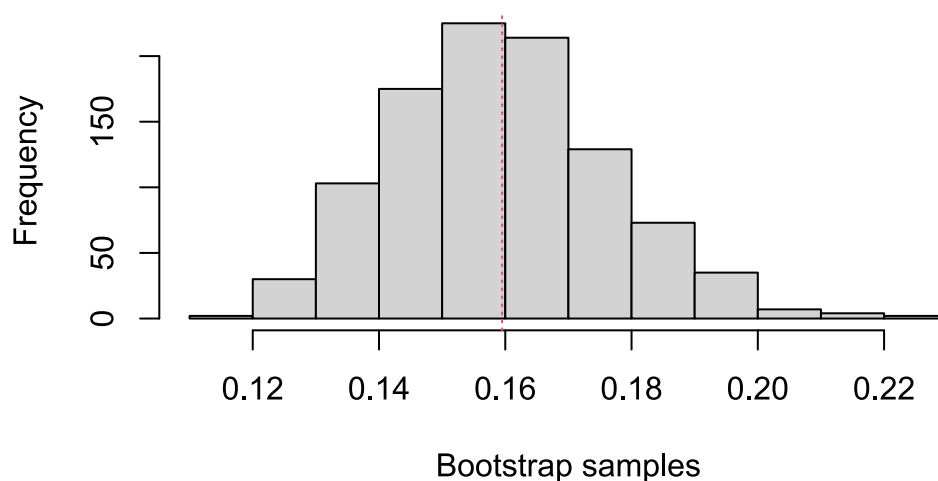


Figure 6.7: Bootstrap distribution of the mean difference in CCC (aided minus unaided)

Using the bootstrap package:

```
library(bootstrap)
b <- bootstrap(sbr$diff, 999, mean)
quantile(b$thetastar, c(.025,.975))
```

```
2.5%    97.5%
0.129975 0.196500
```

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

```
[1] 0.1594615
```

```
# the bootstrap standard error is the SD of the bootstrap distribution
sd(b$thetastar)
```

```
[1] 0.01692876
```

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.

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 the paired differences are normally distributed. (The paired t-test does not require equal variances, because it is based on the differences.) Let's now apply the two tests to our data and compare the results.

```
# normality test of the paired differences
shapiro.test(sbr$diff)
```

Shapiro-Wilk normality test

```
data:  sbr$diff
W = 0.80256, p-value = 0.0009437
```

```
# 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 mean difference is not equal to 0
95 percent confidence interval:
 0.1216158 0.1973842
sample estimates:
mean difference
      0.1595
```

```
# 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
```

The Shapiro-Wilk test indicates that the differences depart from normality ($P < 0.001$), so the non-parametric test is the safer choice. Both tests nevertheless reject the null hypothesis of no difference between the aided and unaided conditions.

Mixed effects modeling

Mixed models extend linear models to data in which the observations are not independent. Here each rater estimated severity twice, once without and once with the SAD, so the estimates of the same rater are correlated (Brown, 2021). A mixed model has *fixed effects*, the regression parameters of interest (here, the effect of using the SAD), and *random effects*, which capture the variation among groups of observations (here, among raters).

We treat raters as a random effect because they are a sample from a larger population of possible raters, and we want conclusions that apply to that population. Accounting for the differences among raters also makes the comparison between the aided and unaided conditions more precise.

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 heteroscedasticity (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
```

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

```
condition response      SE   df lower.CL upper.CL
aided          0.968 0.00359 25.5    0.959    0.974
unaided        0.817 0.01720 25.5    0.779    0.849
```

Degrees-of-freedom method: kenward-roger
 Confidence level used: 0.95
 Intervals are back-transformed from the logit scale

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

```
contrast      odds.ratio      SE df null t.ratio p.value
aided / unaided      6.72 0.597 19    1  21.411 <0.0001
```

Degrees-of-freedom method: kenward-roger
 Tests are performed on the log odds ratio scale

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

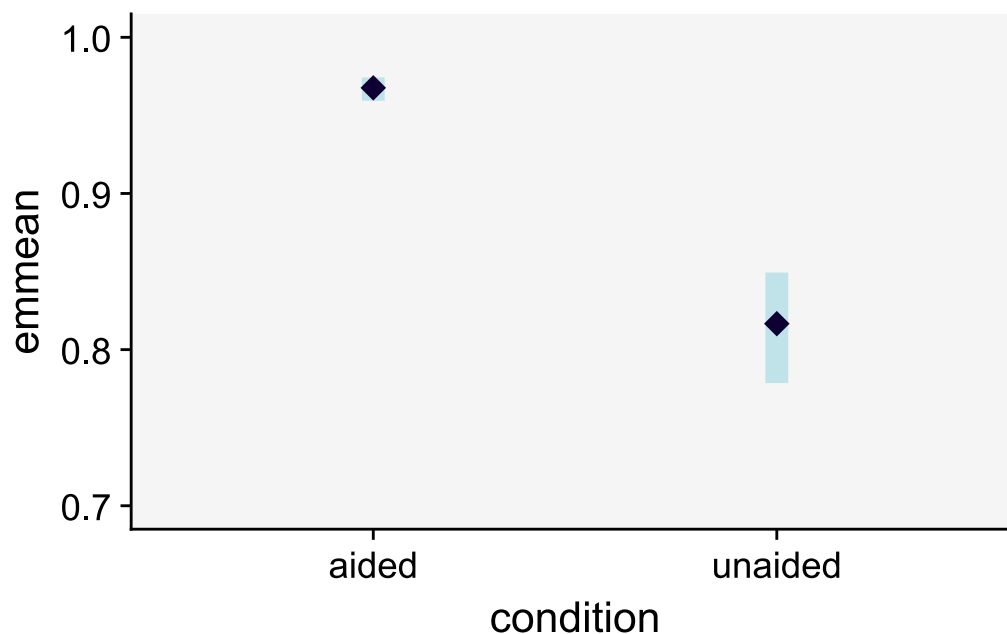


Figure 6.8: Estimated mean CCC and 95% confidence intervals for the unaided and aided conditions, from the mixed model fitted to the logit-transformed CCC

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

Alternatively, we can fit a generalized linear mixed model (GLMM), which allows the response to follow a distribution other than the normal. Because the CCC is bounded between 0 and 1, a beta distribution is a natural choice. The `glmmTMB()` function of the *glmmTMB* package fits GLMMs with a variety of distributions (Brooks et al., 2017). We select the beta distribution with `family = beta_family()`.

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

Residual diagnostics for GLMMs fitted with `glmmTMB()` are best obtained with the *DHARMa* package, which compares standardized residuals with residuals simulated from the fitted model.

```
library(DHARMa)

plot(simulateResiduals(mix2))
```

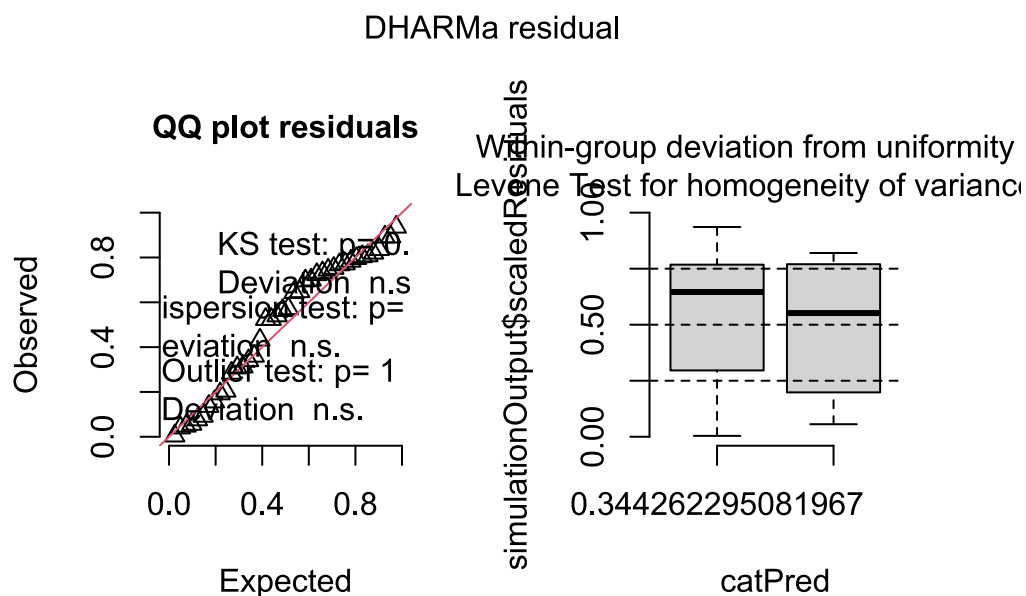


Figure 6.9: Residual diagnostics (DHARMa) of the beta mixed model fitted with `glmmTMB`

The simulated residuals are uniformly distributed, which is what we expect when the model fits the data well. We can now proceed with the post hoc comparison, and we see that the results are similar to those obtained with the logit-transformed response.

```
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, type = "response")
em
```

condition	response	SE	df	asympt.LCL	asympt.UCL
aided	0.967	0.0043	Inf	0.958	0.975
unaided	0.814	0.0167	Inf	0.779	0.845

Confidence level used: 0.95
Intervals are back-transformed from the logit scale

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

contrast	odds.ratio	SE	df	null	z.ratio	p.value
aided / unaided	6.71	0.638	Inf	1	20.023	<0.0001

Tests are performed on the log odds ratio scale

7. Training sessions

Why train raters?

Standard area diagrams (SADs) are not the only way to improve visual estimates of disease severity. Exposing raters to a diverse set of diagrams or images with known severity values also improves their accuracy. Through repeated exposure and immediate feedback, raters calibrate their judgments against the true values and gradually build a reliable mental scale of severity. This process, called *training*, is most useful when visual estimation is the main method of assessment, and it complements other aids such as SADs.

Software

Computerized training has a long history in plant pathology. Programs such as AREAGRAM, DISTRAIN, DISEASE.PRO, ESTIMATE, SEVERITY.PRO and COMBRO appeared in the mid-1980s, soon after personal computers became available, and ran under DOS or Windows. They showed raters computerized images with known, measured severities and gave feedback on each estimate (Bock et al., 2021b).

The benefits of training can be short-lived, because estimation skills fade without regular practice. Raters may therefore need to repeat the training before an assessment to keep their accuracy and precision.

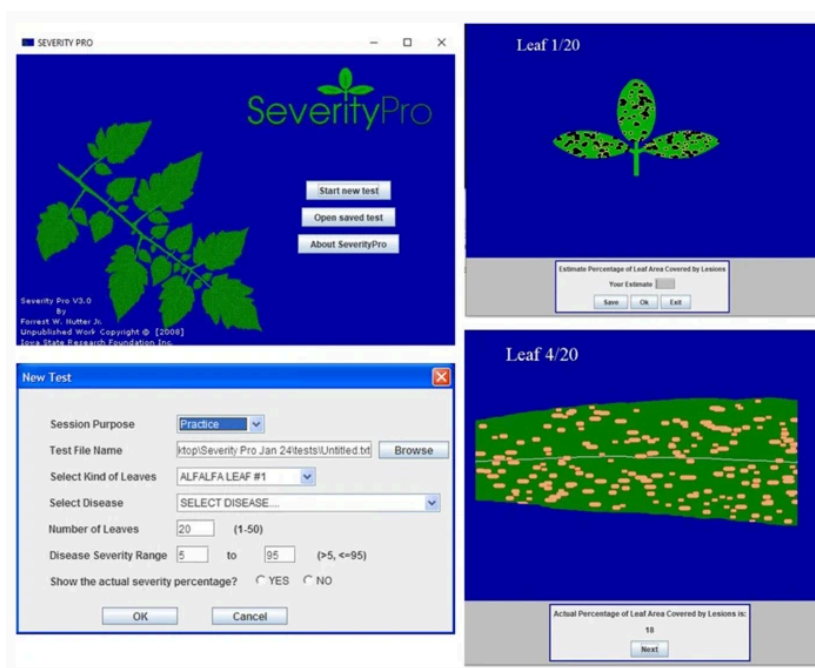


Figure 7.1: Selected screenshots from Severity.Pro, the disease assessment training program by Forrest W. Nutter (Madden et al., 2021).

Online training tools

In Brazil, the “Sistema de treinamento de acuidade visual” began as a web-based system to train raters in assessing citrus canker. It now has a current version for iOS and Android, available at this [link](#).

In Mexico, **Validar-PER** trains raters to assess the severity of coffee leaf rust using diagrammatic log-based scales. It is available online [here](#).

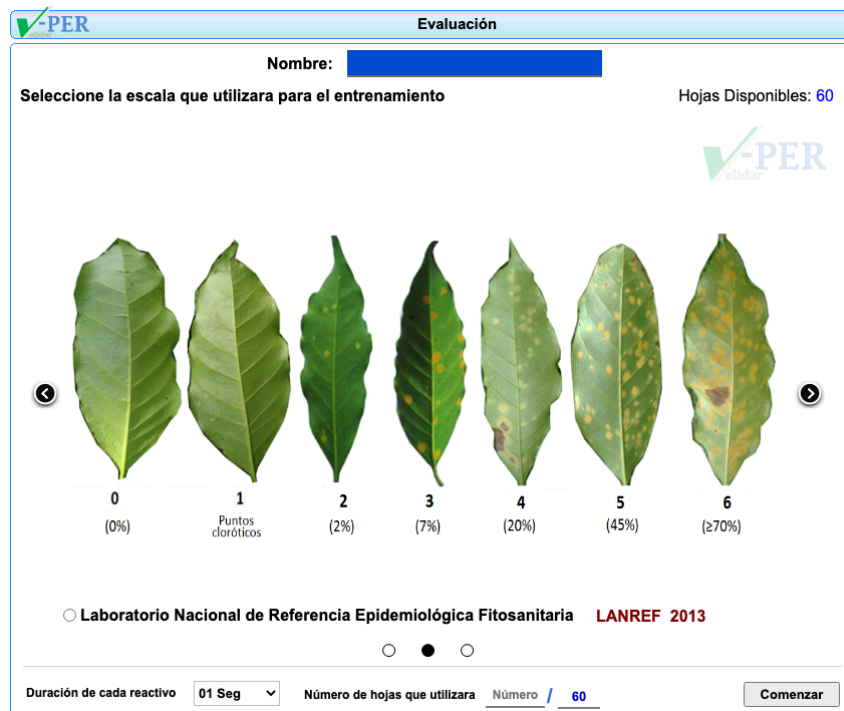


Figure 7.2: Screen of Validar-PER, an online training module for assessing coffee leaf rust severity

Training software made with R

TraineR

TraineR is a Shiny app developed by the author of this book to train users to estimate severity as the percentage area of a leaf or fruit covered by lesions.

Users first set the organ shape and color and the lesion shape, color, number and size, and the app draws an ellipsoidal diagram with these characteristics. They then choose the number of attempts and click “generate new”. For each diagram, the user types an estimate of the diseased area (%), and the app records it in a table next to the actual value.

After the last attempt, the app reports the user’s accuracy as Lin’s concordance correlation coefficient, shows plots of the estimates against the actual values and of the estimation errors, and gives further accuracy statistics.

The app has some limitations: lesions cannot overlap, and the maximum severity is about 60%. It remains a useful educational and demonstration tool.

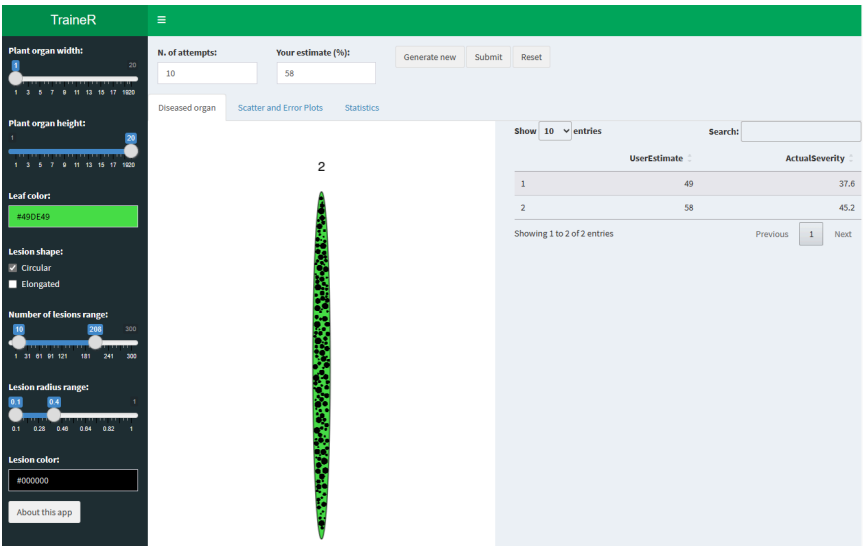


Figure 7.3: Screen of TraineR, an online app for training in the assessment of plant disease severity

Trainer2

Trainer2, the second generation of TraineR, uses photographs of real symptoms instead of simulated diagrams. Users estimate the percentage area affected from the photographs, which makes the training closer to field assessment.

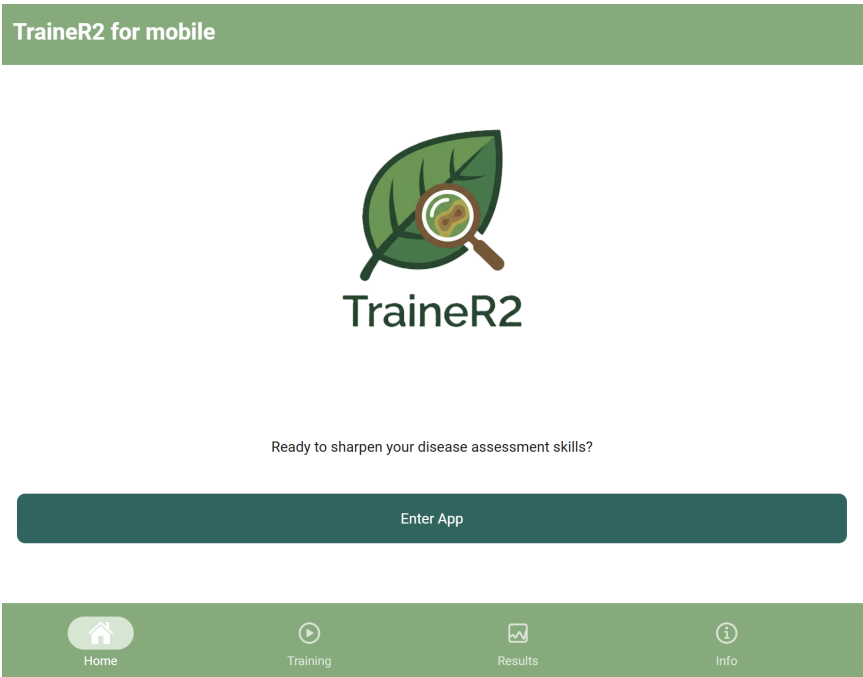


Figure 7.4: Screen of trainer2, an online app for training in the assessment of plant disease severity based on real symptoms captured in photographs

8. Remote sensing

Introduction

Remote sensing techniques, in the sense of gathering & processing of data by a device separated from the object under study, are increasingly providing an important component of the set of technologies available for the study of vegetation systems and their functioning. This is despite the fact that many applications only provide indirect estimations of the biophysical variables of interest (Jones & Vaughan, 2010).

Particular advantages of remote sensing for vegetation studies are that: (i) it is non-contact and non-destructive; and (ii) observations are easily extrapolated to larger scales. Even at the plant scale, remotely sensed imagery is advantageous as it allows rapid sampling of large number of plants (Jones & Vaughan, 2010).

This chapter aims at providing a conceptual & practical approach to apply remote sensing data and techniques to infer information useful for monitoring crop diseases. The structure of this chapter is divided into four sections. The first one introduces basic remote sensing concepts and provides a summary of applications of remote sensing of crop diseases. The second one illustrates a case study focused on identification of banana Fusarium wilt from multispectral UAV imagery. The third one illustrates a case study dealing with estimation of cercospora leaf spot disease on table beet. Finally, it concludes with several reflections about potential and limitations of this technology.

Remote sensing background

Optical remote sensing

Optical remote sensing makes use of the radiation reflected by a surface in the visible (~400-700 nm), the near infrared (700-1300 nm) and shortwave infrared (1300-~3000 nm) parts of the electromagnetic spectrum. Spaceborne & airborne-based remote sensing and field spectroscopy utilize the solar radiation as an illumination source. Lab spectroscopy utilizes a lamp as an artificial illumination source (Figure 8.1).

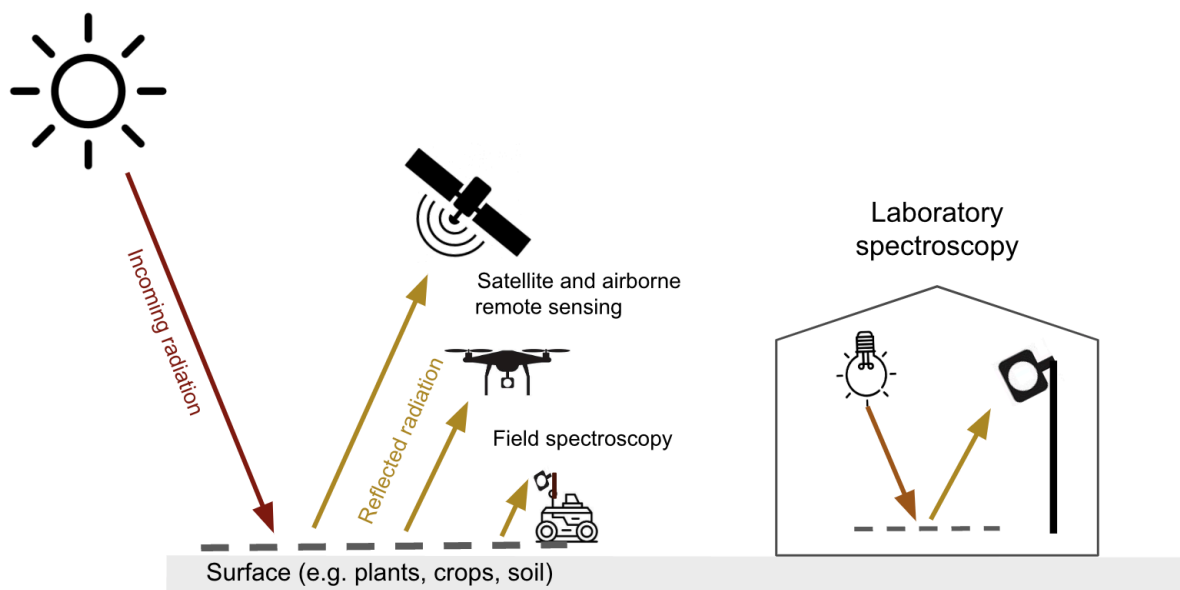


Figure 8.1: Optical remote sensing via spaceborne sensors, field spectroscopy and laboratory spectroscopy (Adapted from https://pages.cms.hu-berlin.de/EOL/geo_rs/index.htm).

The proportion of the radiation reflected by a surface depends on the surface's spectral reflection, absorption and transmission properties and varies with wavelength (Figure 8.2). These spectral properties in turn depend on the surface's physical and chemical constituents (Figure 8.2). Measuring the reflected radiation hence allows us to draw conclusions on a surface's characteristic, which is the basic principle behind optical remote sensing.

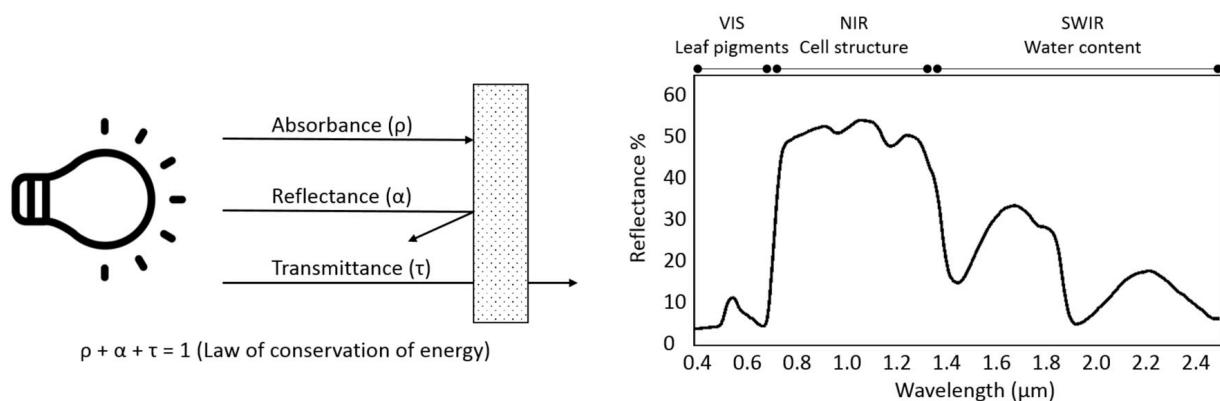


Figure 8.2: Reflection, absorption and transmission by a surface (left). Spectral reflectance profile of a vegetation with major factors determining the reflection (right). Source: https://pages.cms.hu-berlin.de/EOL/geo_rs/

Vegetation spectral properties

Optical remote sensing enables the deduction of various vegetation-related characteristics, including biochemical properties (e.g., pigments, water content), structural properties (e.g., leaf area index (LAI), biomass) or process properties (e.g., light use efficiency (LUE)). The ability to deduce these characteristics depends on the ability of a sensor to resolve vegetation spectra. Hyperspectral sensors capture spectral information in hundreds of narrow and contiguous bands in the VIS, NIR and SWIR, and, thus, resolve subtle absorption features

caused by specific vegetation constituents (e.g. anthocyanins, carotenoids, lignin, cellulose, proteins). In contrast, multispectral sensors capture spectral information in a few broad spectral bands and, thus, only resolve broader spectral features. Still, multispectral systems like Sentinel-2 have been demonstrated to be useful to derive valuable vegetation properties (e.g., LAI, chlorophyll).

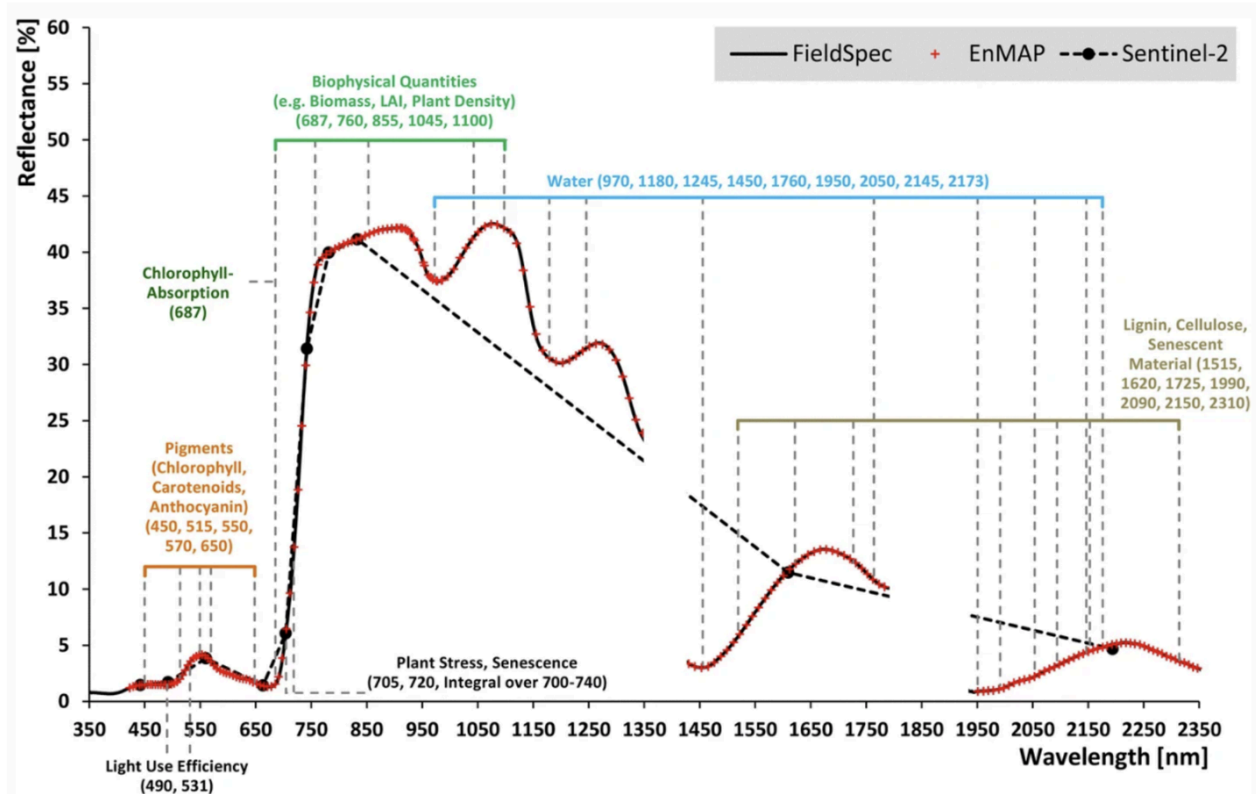


Figure 8.3: Vegetation spectrum in hyperspectral (ASD FieldSpec4, EnMAP) and multispectral (Sentinel-2) resolution as well as characteristic spectral features caused by various constituents and processes (absorption lines shown as grey dashed lines). Source: (Hank et al., 2018)

What does a remote sensor measure?

Optical sensors/spectrometers measure the radiation reflected by a surface to a certain solid angle in the physical quantity radiance. The unit of radiance is watts per square meter per steradian ($\text{W} \cdot \text{m}^{-2} \cdot \text{sr}^{-1}$) (Figure 8.4). In other words, radiance describes the amount of energy (W) that is reflected from a surface (m^{-2}) and arrives at the sensor in a three-dimensional angle (sr^{-1}).

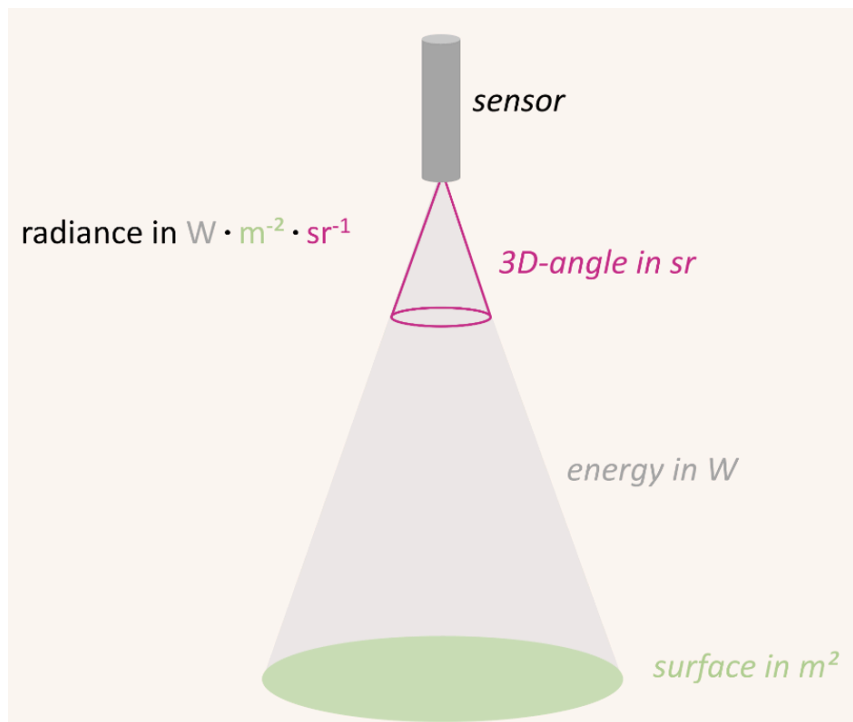


Figure 8.4: Source: https://pages.cms.hu-berlin.de/EOL/geo_rs/

A general problem related to the use of radiance as unit of measurement is the variation of radiance values with illumination. For example, the absolute incoming solar radiation varies over the course of the day as a function of the relative position between sun and surface and so does the absolute amount of radiance measured. We can only compare measurements taken a few hours apart or on different dates when we are putting the measured radiance in relation to the incoming illumination.

The quotient between measured reflected radiance and measured incoming radiance ($\text{Radiance}_{\text{reflected}} / \text{Radiance}_{\text{incoming}}$) is called reflectance (usually denoted as ρ). Reflectance provides a stable unit of measurement which is independent from illumination and is the percentage of the total measurable radiation, which has not been absorbed or transmitted.

Hyperspectral vs. multispectral imagery

Hyperspectral imaging involves capturing and analyzing data from a large number of narrow, contiguous bands across the electromagnetic spectrum, resulting in a high-resolution spectrum for each pixel in the image. As a result, a hyperspectral camera provides smooth spectra. The spectra provided by multispectral cameras are more like stairs or saw teeth without the ability to depict acute spectral signatures (Figure 8.6).

Vegetation Indices

A vegetation index (VI) represents a spectral transformation of two or more bands of spectral imagery into a single-band image. A VI is designed to enhance the vegetation signal with regard to different vegetation properties, while minimizing confounding factors such as soil background reflectance, directional, or atmospheric effects. There are many different VIs, including multispectral broadband indices as well as hyperspectral narrowband indices.

Most of the multispectral broadband indices make use of the inverse relationship between the lower reflectance in the red (through chlorophyll absorption) and higher reflectance in the

near-infrared (through leaf structure) to provide a measure of greenness that can be indirectly related to biochemical or structural vegetation properties (e.g., chlorophyll content, LAI). The Normalized Difference Vegetation Index (NDVI) is one of the most commonly used broadband VIs:

$$NDVI = \frac{\rho_{nir} - \rho_{red}}{\rho_{nir} + \rho_{red}} \quad (8.1)$$

The interpretation of the absolute value of the NDVI is highly informative, as it allows the immediate recognition of the areas of the farm or field that have problems. The NDVI is a simple index to interpret: its values vary between -1 and 1 , and each value corresponds to a different agronomic situation, regardless of the crop (Figure 8.5)

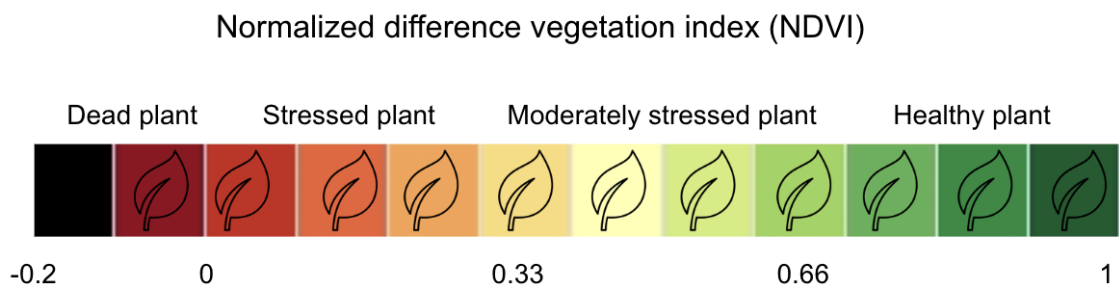


Figure 8.5: Agronomic conditions depending on the values in a NDVI scale

Remote sensing of crop diseases

Detection of plant stress

One popular use of remote sensing is in diagnosis and monitoring of plant *responses* to biotic (i.e. disease and insect damage) and abiotic stress (e.g. water stress, heat, high light, pollutants) with hundreds of publications on the topic. It is worth noting that most available techniques monitor the plant *response* rather than the stress itself. For example, with some diseases, it is common to estimate changes in canopy cover (using vegetation indices) as measures of “disease” but this measure could also be associated to water deficit (Jones & Vaughan, 2010). This highlights the importance of measuring crop conditions in the field & laboratory to collect reliable data and be able to disentangle complex plant responses. Anyway, remote sensing can be used as the first step in site-specific disease control and also to phenotype the reactions of plant genotypes to pathogen attack (Lowe et al., 2017).

Optical methods for measuring crop disease

There are a variety of optical sensors for the assessment of plant diseases. Sensors can be based only on the visible spectrum (400-700 nm) or on the visible and/or infrared spectrum (700 nm - 1mm). The latter may include near-infrared (NIR) (0.75-1.4 μm), short wavelength infrared (SWIR) (1.4-3 μm), medium wavelength infrared (MWIR) (3-8 μm), or thermal infrared (8-15 μm) (Figure 8.6). Sensors record either imaging or non imaging (i.e average) spectral radiance values which need to be converted to reflectance before conducting any crop disease monitoring task.

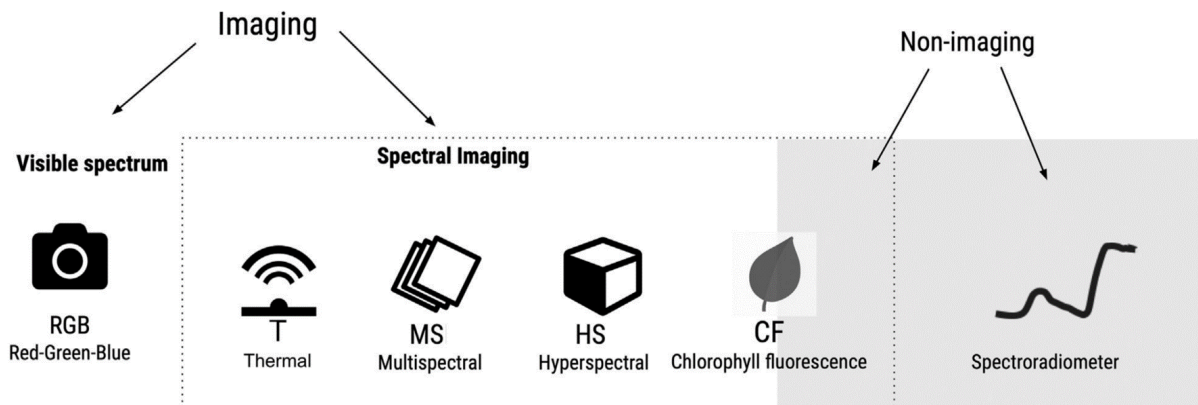


Figure 8.6: source: Del Ponte et al. (2024)

In a recent chapter of Agrios' Plant Pathology, Del Ponte et al. (2024) highlights the importance of understanding the basic principles of the interaction of light with plant tissue or the plant canopy as a crucial prerequisite for the analysis and interpretation for disease assessment. When a plant is infected, there are changes to the physiology and biochemistry of the host, with the eventual development of disease symptoms and/or signs of the pathogen which may be accompanied by structural and biochemical changes that affect absorbance, transmittance, and reflectance of light (Figure 8.7).

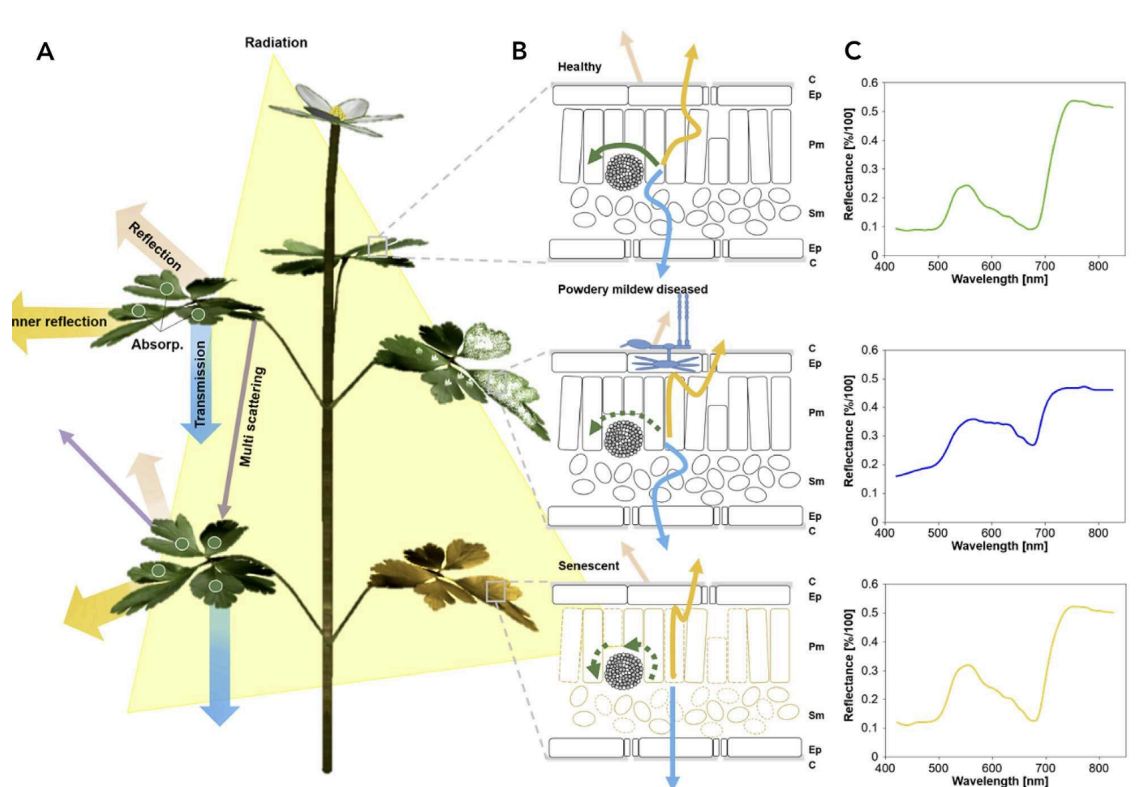


Figure 8.7: source: Del Ponte et al. (2024)

Scopes of disease sensing

The quantification of typical disease symptoms (disease severity) and assessment of leaves infected by several pathogens are relatively simple for imaging systems but may become a

challenge for nonimaging sensors and sensors with inadequate spatial resolution (Oerke, 2020). Systematic monitoring of a crop by remote sensors can allow farmers to take preventive actions if infections are detected early. Remote sensing sensors & processing techniques need to be carefully selected to be capable of (a) detecting a deviation in the crop's health status brought about by pathogens, (b) identifying the disease, and (c) quantifying the severity of the disease. Remote sensing can also be effectively used in (d) food quality control (Figure 8.8).

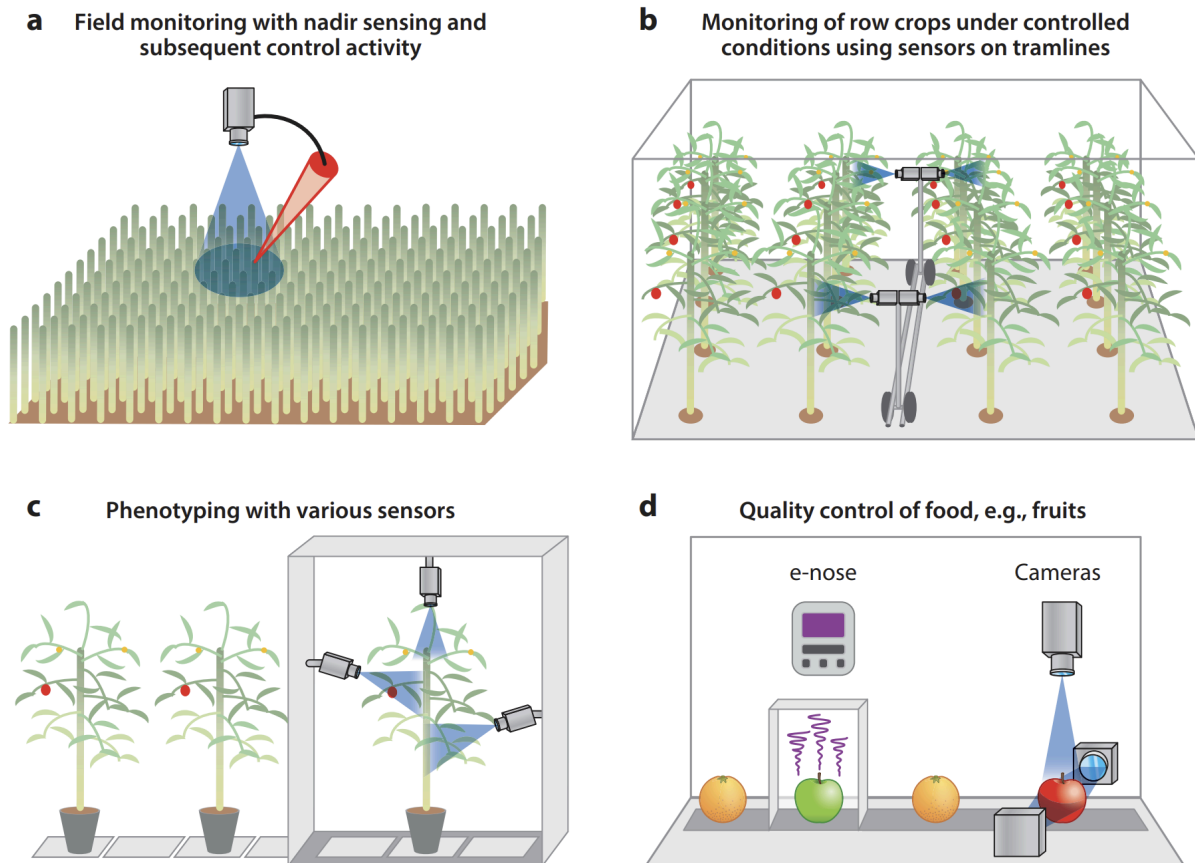


Figure 8.8: Source: (Oerke, 2020)

Monitoring plant diseases

Sensing of plants for precision disease control is done in large fields or greenhouses where the aim is to detect the occurrence of diseases at the early stages of epidemics, i.e., at low symptom frequency. Lowe et al. (2017) reviewed hyperspectral imaging of plant diseases, focusing on early detection of diseases for crop monitoring. They report several analysis techniques successfully used for the detection of biotic and abiotic stresses with reported levels of accuracy higher than 80%.

Technique	Plant (stress)
Quadratic discriminant analysis (QDA)	Wheat (yellow rust)
	Avocado (laurel wilt)
Decision tree (DT)	Avocado (laurel wilt)
	Sugarbeet (cercospora leaf spot)

Technique	Plant (stress)
	Sugarbeet (powdery mildew)
	Sugarbeet (leaf rust)
	Wheat (yellow rust)
Partial least squares regression (PLSR)	Celery (sclerotinia rot)
Raw	
Savitzky-Golay 1st derivative	
Savitzky-Golay 2nd derivative	
Partial least squares regression (PLSR)	Wheat (yellow rust)
Fisher's linear discriminant analysis	Wheat (aphid)
	Wheat (powdery mildew)
	Wheat (powdery mildew)
	Cucumber (downy mildew)
Spectral angle mapper (SAM)	Sugarbeet (cercospora leaf spot)
	Sugarbeet (powdery mildew)
	Sugarbeet (leaf rust)
	Wheat (head blight)
	Sugarbeet (cercospora leaf spot)
Artificial neural network (ANN)	Sugarbeet (powdery mildew)
	Sugarbeet (leaf rust)
	Sugarbeet (cercospora leaf spot)
Support vector machine (SVM)	Sugarbeet (powdery mildew)
	Sugarbeet (leaf rust)
	Barley (drought)
	Grapefruit
Spectral information divergence (SID)	(canker, greasy spot, insect
	damage, scab, wind scar)

Lowe et al. (2017) state that remote sensing of diseases under production conditions is challenging because of variable environmental factors and crop-intrinsic characteristics, e.g., 3D architecture, various growth stages, variety of diseases that may occur simultaneously, and the high sensitivity required to reliably perceive low disease levels suitable for decision-making in disease control. The use of less sensitive systems may be restricted to the assessment of crop damage and yield losses due to diseases.

UAV applications for plant disease detection and monitoring

Kouadio et al. (2023) undertook a systematic quantitative literature review to summarize existing literature in UAV-based applications for plant disease detection and monitoring. Results reveal a global disparity in research on the topic, with Asian countries being the top contributing countries. World regions such as Oceania and Africa exhibit comparatively lesser representation. To date, research has largely focused on diseases affecting wheat, sugar beet, potato, maize, and grapevine (Figure 8.9). Multispectral, red-green-blue, and hyperspectral

sensors were most often used to detect and identify disease symptoms, with current trends pointing to approaches integrating multiple sensors and the use of machine learning and deep learning techniques. The authors suggest that future research should prioritize (i) development of cost-effective and user-friendly UAVs, (ii) integration with emerging agricultural technologies, (iii) improved data acquisition and processing efficiency (iv) diverse testing scenarios, and (v) ethical considerations through proper regulations.



Figure 8.9: Source: Kouadio et al. (2023)

Disease detection

This section illustrates the use of unmanned aerial vehicle (UAV) remote sensing imagery for identifying banana wilt disease. Fusarium wilt of banana, also known as "banana cancer", threatens banana production areas worldwide. Timely and accurate identification of Fusarium wilt disease is crucial for effective disease control and optimizing agricultural planting structure (Pegg et al., 2019).

A common initial symptom of this disease is the appearance of a faint pale yellow streak at the base of the petiole of the oldest leaf. This is followed by leaf chlorosis which progresses from lower to upper leaves, wilting of leaves and longitudinal splitting of their bases. Pseudostem splitting of leaf bases is more common in young, rapidly growing plants Pegg et al. (2019).



Figure 8.10: Cavendish plant affected by Race 1 Foc (D. Peasley). Source: (Pegg et al., 2019)

Ye et al. (2020) made publicly available experimental data (Huichun YE et al., 2022) on wilted banana plants collected in a banana plantation located in Long'an County, Guangxi (China). The data set includes UAV multispectral reflectance data and ground survey data on the incidence of banana wilt disease. The paper by Ye et al. (2020) reports that the banana Fusarium wilt disease can be easily identified using several vegetation indices (VIs) obtained from this data set. Tested VIs include green chlorophyll index (CIgreen), red-edge chlorophyll index (CI_{RE}), normalized difference vegetation index (NDVI), and normalized difference red-edge index (NDRE). The dataset can be downloaded from [here](#).

Software setup

Let's start by cleaning up R memory:

```
rm(list=ls())
```

Then, we need to install several packages (if they are not installed yet):

```
list.of.packages <- c("terra",  
                      "tidyterra",
```

```

      "stars",
      "sf",
      "leaflet",
      "leafem",
      "dplyr",
      "ggplot2",
      "tidymodels")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)

```

Now, let's load all the required packages:

```

library(terra)
library(tidyterra)
library(stars)
library(sf)
library(leaflet)
library(leafem)
library(dplyr)
library(ggplot2)
library(tidymodels)

```

Reading the dataset

The following code assumes you have already downloaded the [Huichun YE et al. \(2022\)](#) dataset and unzipped its content under the *data/banana_data* directory.

File formats

Let's list the files under each subfolder:

```
list.files("data/banana_data/1_UAV multispectral reflectance")
```

```

[1] "UAV multispectral reflectance.tfw"
[2] "UAV multispectral reflectance.tif"
[3] "UAV multispectral reflectance.tif.aux.xml"
[4] "UAV multispectral reflectance.tif.ovr"

```

Note that the *.tif* file contains an orthophotomosaic of surface reflectance. It was created from UAV images taken with a *Micasense Red Edge M* camera which has five narrow spectral bands: Blue (465–485 nm), green (550–570 nm), red (653–673 nm), red edge (712–722 nm), and near-infrared (800–880 nm). We assume here that those images have been radiometrically and geometrically corrected.

```
list.files("data/banana_data/2_Ground survey data of banana Fusarium wilt")
```

```

[1] "Ground_survey_data_of_banana_Fusarium_wilt.dbf"
[2] "Ground_survey_data_of_banana_Fusarium_wilt.prj"
[3] "Ground_survey_data_of_banana_Fusarium_wilt.sbn"
[4] "Ground_survey_data_of_banana_Fusarium_wilt.sbx"
[5] "Ground_survey_data_of_banana_Fusarium_wilt.shp"
[6] "Ground_survey_data_of_banana_Fusarium_wilt.shp.xml"
[7] "Ground_survey_data_of_banana_Fusarium_wilt.shx"

```

This is a shapefile with 80 points where the plant health status was collected on the same date as the images.

```
list.files("data/banana_data/3_Boundary of banana planting region")
```

```
[1] "Boundary_of_banana_planting_region.dbf"
[2] "Boundary_of_banana_planting_region.prj"
[3] "Boundary_of_banana_planting_region.sbn"
[4] "Boundary_of_banana_planting_region.sbx"
[5] "Boundary_of_banana_planting_region.shp"
[6] "Boundary_of_banana_planting_region.shp.xml"
[7] "Boundary_of_banana_planting_region.shx"
```

This is a shapefile with one polygon representing the boundary of the study area.

Read the orthomosaic and the ground data

Now, let's read the orthomosaic using the *terra* package:

```
# Open the tif
tif <- "data/banana_data/1_UAV multispectral reflectance/UAV multispectral reflectance.tif"

rrr <- terra::rast(tif)
```

Let's check what we get:

```
rrr
```

```
class      : SpatRaster
size       : 7885, 14420, 5  (nrow, ncol, nlyr)
resolution : 0.08, 0.08  (x, y)
extent     : 779257.9, 780411.5, 2560496, 2561127  (xmin, xmax, ymin, ymax)
coord. ref.: WGS 84 / UTM zone 48N (EPSG:32648)
source     : UAV multispectral reflectance.tif
names      : UAV mu~ance_1, UAV mu~ance_2, UAV mu~ance_3, UAV mu~ance_4, UAV mu~ance_5
min values :      0.000000,      0.000000,      0.000000,      0.000000,      0.000000
max values :      1.272638,      1.119109,      1.075701,      0.9651694,      1.069767
```

Note that this is a 5-band multispectral image with 8 cm pixel size.

Now, let's read the ground data:

```
shp <- "data/banana_data/2_Ground survey data of banana Fusarium wilt/
Ground_survey_data_of_banana_Fusarium_wilt.shp"
ggg <- sf::st_read(shp)
```

```
Reading layer `Ground_survey_data_of_banana_Fusarium_wilt' from data source
`C:\Users\edelp\OneDrive\Documents\GitHub\epidemiology-R_review\data\banana_data\2_Ground survey data of
banana Fusarium wilt\Ground_survey_data_of_banana_Fusarium_wilt.shp'
using driver `ESRI Shapefile'
Simple feature collection with 80 features and 4 fields
Geometry type: POINT
Dimension: XY
Bounding box: xmin: 779548.9 ymin: 2560702 xmax: 780097 ymax: 2561020
Projected CRS: WGS 84 / UTM zone 48N
```

What we got?

```
ggg
```

```
Simple feature collection with 80 features and 4 fields
Geometry type: POINT
Dimension:      XY
Bounding box:   xmin: 779548.9 ymin: 2560702 xmax: 780097 ymax: 2561020
Projected CRS: WGS 84 / UTM zone 48N
First 10 features:
  OBJECTID  样点类型   x_经度   y_纬度      geometry
1         1  健康植株 107.7326 23.13240 POINT (779838.5 2560800)
2         2  健康植株 107.7332 23.13316 POINT (779901.2 2560885)
3         3  健康植株 107.7334 23.13394 POINT (779920.1 2560971)
4         4  健康植株 107.7326 23.13430 POINT (779837.5 2561010)
5         5  健康植株 107.7302 23.13225 POINT (779595.2 2560779)
6         6  健康植株 107.7301 23.13190 POINT (779584.6 2560739)
7         7  健康植株 107.7300 23.13297 POINT (779569.6 2560857)
8         8  健康植株 107.7315 23.13301 POINT (779729.4 2560865)
9         9  健康植株 107.7313 23.13245 POINT (779710.5 2560803)
10        10  健康植株 107.7349 23.13307 POINT (780078.9 2560879)
```

Note that the attributes are in Chinese language. It seems that we will need to do several changes.

Visualizing the data

As the orthomosaic is too heavy to visualize, we will need a coarser version of it. Let's use the *terra* package for doing it.

```
rrr8 <- terra::aggregate(rrr, 8)
```

```
|-----|-----|-----|-----|
=====
```

```
#terra <- resample(elev, template, method='bilinear')
```

Let's check the output:

```
rrr8
```

```
class      : SpatRaster
size       : 986, 1803, 5  (nrow, ncol, nlyr)
resolution : 0.64, 0.64  (x, y)
extent     : 779257.9, 780411.8, 2560496, 2561127  (xmin, xmax, ymin, ymax)
coord. ref.: WGS 84 / UTM zone 48N (EPSG:32648)
source(s)  : memory
names      : UAV mu~ance_1, UAV mu~ance_2, UAV mu~ance_3, UAV mu~ance_4, UAV mu~ance_5
min values :      0.000000,      0.000000,      0.000000,      0.000000,      0.000000
max values :      1.272638,      1.119109,      1.075701,      0.949925,      1.069767
```

Note that the pixel size of the aggregated raster is 64 cm. Now, in order to visualize the ground points, we will need a color palette:


```
pal <- colorFactor(
  palette = c('green', 'red'),
  domain = ggg$ 样点类型
)
```

Then, we will use the *leaflet* package to plot the new image and the ground points:

i Interactive content

The map of the image and the ground points is interactive and only available in the online edition of the book: <https://r4pde.net/data-remote-sensing.html>.

Extracting image values at sampled points

Now we will extract raster values at point locations using the `st_extract()` function from the {stars} library. It is expected that a value per band is extracted at each point.

We need to convert the *raster* object into a *stars* object:

```
sss <- st_as_stars(rrr)
```

What we got?

```
sss
```

```
stars_proxy object with 1 attribute in 1 file(s):
$`UAV multispectral reflectance.tif`
[1] "[...]/UAV multispectral reflectance.tif"

dimension(s):
      from   to offset delta      refsys point x/y
x       1 14420 779258 0.08 WGS 84 / UTM zone 48N FALSE [x]
y       1  7885 2561127 -0.08 WGS 84 / UTM zone 48N FALSE [y]
band    1    5    NA    NA           NA    NA
```

Before conducting the extraction task, it is advisable to collect band values not at a single pixel but at a small window (e.g. 3×3 pixels). Thus, we will start creating 20cm buffers at each site:

```
poly <- st_buffer(ggg, dist = 0.20)
```

Now, the extraction task:

```
# Extract the median value per polygon
buf_values <- aggregate(sss, poly, FUN = median) |>
  st_as_sf()
```

What we got:

```
buf_values
```

```
Simple feature collection with 80 features and 5 fields
Geometry type: POLYGON
Dimension:      XY
Bounding box:   xmin: 779548.7 ymin: 2560702 xmax: 780097.2 ymax: 2561021
```

```

Projected CRS: WGS 84 / UTM zone 48N
First 10 features:
  UAV multispectral reflectance.tif.V1 UAV multispectral reflectance.tif.V2
1          0.2837147          0.3570867
2          0.3096394          0.4598360
3          0.2652082          0.3556182
4          0.2729177          0.3626878
5          0.3033864          0.3089822
6          0.2993084          0.3655908
7          0.2471483          0.2859972
8          0.2491679          0.3223873
9          0.3213297          0.4069843
10         0.2976966          0.3430840
  UAV multispectral reflectance.tif.V3 UAV multispectral reflectance.tif.V4
1          0.2049306          0.6370987
2          0.2058005          0.6609572
3          0.1769855          0.5988070
4          0.1828043          0.7094992
5          0.1894684          0.6805743
6          0.1988409          0.7750074
7          0.1880568          0.5221797
8          0.1855947          0.5934904
9          0.2013687          0.5593004
10         0.2144508          0.8057290
  UAV multispectral reflectance.tif.V5          geometry
1          0.4779058 POLYGON ((779838.7 2560800,...
2          0.5714727 POLYGON ((779901.4 2560885,...
3          0.4352523 POLYGON ((779920.3 2560971,...
4          0.5149931 POLYGON ((779837.7 2561010,...
5          0.4008912 POLYGON ((779595.4 2560779,...
6          0.4967270 POLYGON ((779584.8 2560739,...
7          0.3308629 POLYGON ((779569.8 2560857,...
8          0.4570443 POLYGON ((779729.6 2560865,...
9          0.5094757 POLYGON ((779710.7 2560803,...
10         0.5005793 POLYGON ((780079.1 2560879,...

```

Note that names of bands are weird:

```
names(buf_values)
```

```

[1] "UAV multispectral reflectance.tif.V1"
[2] "UAV multispectral reflectance.tif.V2"
[3] "UAV multispectral reflectance.tif.V3"
[4] "UAV multispectral reflectance.tif.V4"
[5] "UAV multispectral reflectance.tif.V5"
[6] "geometry"

```

Let's rename band values:

```

buf_values |> rename(blue = "UAV multispectral reflectance.tif.V1",
                    green = "UAV multispectral reflectance.tif.V2",
                    red = "UAV multispectral reflectance.tif.V3",
                    redge = "UAV multispectral reflectance.tif.V4",
                    nir = "UAV multispectral reflectance.tif.V5") -> buf_values2

```

Now, we got shorter names per band:

buf_values2

```
Simple feature collection with 80 features and 5 fields
Geometry type: POLYGON
Dimension:      XY
Bounding box:   xmin: 779548.7 ymin: 2560702 xmax: 780097.2 ymax: 2561021
Projected CRS: WGS 84 / UTM zone 48N
First 10 features:
      blue      green      red      redge      nir
1  0.2837147  0.3570867  0.2049306  0.6370987  0.4779058
2  0.3096394  0.4598360  0.2058005  0.6609572  0.5714727
3  0.2652082  0.3556182  0.1769855  0.5988070  0.4352523
4  0.2729177  0.3626878  0.1828043  0.7094992  0.5149931
5  0.3033864  0.3089822  0.1894684  0.6805743  0.4008912
6  0.2993084  0.3655908  0.1988409  0.7750074  0.4967270
7  0.2471483  0.2859972  0.1880568  0.5221797  0.3308629
8  0.2491679  0.3223873  0.1855947  0.5934904  0.4570443
9  0.3213297  0.4069843  0.2013687  0.5593004  0.5094757
10 0.2976966  0.3430840  0.2144508  0.8057290  0.5005793

      geometry
1  POLYGON ((779838.7 2560800,...
2  POLYGON ((779901.4 2560885,...
3  POLYGON ((779920.3 2560971,...
4  POLYGON ((779837.7 2561010,...
5  POLYGON ((779595.4 2560779,...
6  POLYGON ((779584.8 2560739,...
7  POLYGON ((779569.8 2560857,...
8  POLYGON ((779729.6 2560865,...
9  POLYGON ((779710.7 2560803,...
10 POLYGON ((780079.1 2560879,...
```

Computing vegetation indices

Ye et al. (2020) used the following indices (Figure 8.11):

Table 1. List of eight vegetation indices (Vis) used in this study.

Vegetation Index	Description	Formulation	Sensitive Parameter	Reference
NDVI	Normalized difference vegetation index	$(R_{NIR} - R_{red}) / (R_{NIR} + R_{red})$	Leaf area index, green biomass	[35]
NDRE	Normalized difference red edge index	$(R_{NIR} - R_{RE}) / (R_{NIR} + R_{RE})$	Leaf area index, green biomass	[36]
Cl _{green}	Green chlorophyll index	$R_{NIR} / R_{green} - 1$	Chlorophyll content	[37]
Cl _{RE}	Red-edge chlorophyll index	$R_{NIR} / R_{RE} - 1$	Chlorophyll content	[38]
SIPI	Structural independent pigment index	$(R_{NIR} - R_{blue}) / (R_{NIR} - R_{red})$	Pigment content	[39]
SIPI _{RE}	Red-edge structural independent pigment index	$(R_{RE} - R_{blue}) / (R_{RE} - R_{red})$	Pigment content	[40]
CARI	Carotenoid index	$R_{RE} / R_{green} - 1$	Carotenoid content	[41]
ARI	Anthocyanin reflectance index	$1 / R_{green} - 1 / R_{RE}$	Anthocyanin content	[42]

Figure 8.11

Thus, we will compute several of those indices:

```
buf_indices <- buf_values2 |>
  mutate(ndvi = (nir - red) / (nir+red),
         ndre = (nir - redge) / (nir+redge),
         cire = (nir / redge) - 1,
         sipi = (nir - blue) / (nir - red)
  ) |> select(ndvi, ndre, cire, sipi)
```

What we got:

```
buf_indices
```

```
Simple feature collection with 80 features and 4 fields
Geometry type: POLYGON
Dimension:      XY
Bounding box:   xmin: 779548.7 ymin: 2560702 xmax: 780097.2 ymax: 2561021
Projected CRS: WGS 84 / UTM zone 48N
First 10 features:
   ndvi    ndre    cire    sipi      geometry
1  0.3997667 -0.14277326 -0.24987155 0.7113871 POLYGON ((779838.7 2560800,...
2  0.4704552 -0.07260820 -0.13538624 0.7160328 POLYGON ((779901.4 2560885,...
3  0.4218407 -0.15816766 -0.27313430 0.6584046 POLYGON ((779920.3 2560971,...
4  0.4760534 -0.15884638 -0.27414570 0.7287284 POLYGON ((779837.7 2561010,...
5  0.3581253 -0.25861496 -0.41095167 0.4611839 POLYGON ((779595.4 2560779,...
```

```

6 0.4282631 -0.21881961 -0.35906808 0.6627319 POLYGON ((779584.8 2560739,...
7 0.2751988 -0.22427574 -0.36638109 0.5862114 POLYGON ((779569.8 2560857,...
8 0.4223983 -0.12988254 -0.22990450 0.7658009 POLYGON ((779729.6 2560865,...
9 0.4334381 -0.04661848 -0.08908399 0.6106513 POLYGON ((779710.7 2560803,...
10 0.4001629 -0.23359699 -0.37872496 0.7090615 POLYGON ((780079.1 2560879,...

```

Note that the health status is missing in *buf_indices*. Therefore, we will need to use a spatial join to link such status:

```

samples <- st_join(
  ggg,
  buf_indices,
  join = st_intersects)

```

Let's check the output:

```
samples
```

```

Simple feature collection with 80 features and 8 fields
Geometry type: POINT
Dimension: XY
Bounding box: xmin: 779548.9 ymin: 2560702 xmax: 780097 ymax: 2561020
Projected CRS: WGS 84 / UTM zone 48N
First 10 features:

```

	OBJECTID	样点类型	x_经度	y_纬度	ndvi	ndre	cire
1	1	健康植株	107.7326	23.13240	0.3997667	-0.14277326	-0.24987155
2	2	健康植株	107.7332	23.13316	0.4704552	-0.07260820	-0.13538624
3	3	健康植株	107.7334	23.13394	0.4218407	-0.15816766	-0.27313430
4	4	健康植株	107.7326	23.13430	0.4760534	-0.15884638	-0.27414570
5	5	健康植株	107.7302	23.13225	0.3581253	-0.25861496	-0.41095167
6	6	健康植株	107.7301	23.13190	0.4282631	-0.21881961	-0.35906808
7	7	健康植株	107.7300	23.13297	0.2751988	-0.22427574	-0.36638109
8	8	健康植株	107.7315	23.13301	0.4223983	-0.12988254	-0.22990450
9	9	健康植株	107.7313	23.13245	0.4334381	-0.04661848	-0.08908399
10	10	健康植株	107.7349	23.13307	0.4001629	-0.23359699	-0.37872496

```

      sipi      geometry
1 0.7113871 POINT (779838.5 2560800)
2 0.7160328 POINT (779901.2 2560885)
3 0.6584046 POINT (779920.1 2560971)
4 0.7287284 POINT (779837.5 2561010)
5 0.4611839 POINT (779595.2 2560779)
6 0.6627319 POINT (779584.6 2560739)
7 0.5862114 POINT (779569.6 2560857)
8 0.7658009 POINT (779729.4 2560865)
9 0.6106513 POINT (779710.5 2560803)
10 0.7090615 POINT (780078.9 2560879)

```

It seems we succeeded.

```
unique(samples$样点类型)
```

```
[1] "健康植株" "枯萎病植株"
```

Let's check it:

samples

```
Simple feature collection with 80 features and 8 fields
Geometry type: POINT
Dimension:      XY
Bounding box:   xmin: 779548.9 ymin: 2560702 xmax: 780097 ymax: 2561020
Projected CRS: WGS 84 / UTM zone 48N
First 10 features:
```

	OBJECTID	样点类型	x_经度	y_纬度	ndvi	ndre	cire
1	1	健康植株	107.7326	23.13240	0.3997667	-0.14277326	-0.24987155
2	2	健康植株	107.7332	23.13316	0.4704552	-0.07260820	-0.13538624
3	3	健康植株	107.7334	23.13394	0.4218407	-0.15816766	-0.27313430
4	4	健康植株	107.7326	23.13430	0.4760534	-0.15884638	-0.27414570
5	5	健康植株	107.7302	23.13225	0.3581253	-0.25861496	-0.41095167
6	6	健康植株	107.7301	23.13190	0.4282631	-0.21881961	-0.35906808
7	7	健康植株	107.7300	23.13297	0.2751988	-0.22427574	-0.36638109
8	8	健康植株	107.7315	23.13301	0.4223983	-0.12988254	-0.22990450
9	9	健康植株	107.7313	23.13245	0.4334381	-0.04661848	-0.08908399
10	10	健康植株	107.7349	23.13307	0.4001629	-0.23359699	-0.37872496

```

      sipi      geometry
1 0.7113871 POINT (779838.5 2560800)
2 0.7160328 POINT (779901.2 2560885)
3 0.6584046 POINT (779920.1 2560971)
4 0.7287284 POINT (779837.5 2561010)
5 0.4611839 POINT (779595.2 2560779)
6 0.6627319 POINT (779584.6 2560739)
7 0.5862114 POINT (779569.6 2560857)
8 0.7658009 POINT (779729.4 2560865)
9 0.6106513 POINT (779710.5 2560803)
10 0.7090615 POINT (780078.9 2560879)

```

Now, we will replace the Chinese words for English words:

```
samples |>
  mutate(score = ifelse(样点类型 == '健康植株', 'healthy', 'wilted')) |>
  rename(east = x_经度,
         north = y_纬度 ) |>
  select(OBJECTID,score, ndvi, ndre, cire, sipi) -> nsamples
```

Let's check the output:

nsamples

```
Simple feature collection with 80 features and 6 fields
Geometry type: POINT
Dimension:      XY
Bounding box:   xmin: 779548.9 ymin: 2560702 xmax: 780097 ymax: 2561020
Projected CRS: WGS 84 / UTM zone 48N
First 10 features:
```

	OBJECTID	score	ndvi	ndre	cire	sipi
1	1	healthy	0.3997667	-0.14277326	-0.24987155	0.7113871
2	2	healthy	0.4704552	-0.07260820	-0.13538624	0.7160328
3	3	healthy	0.4218407	-0.15816766	-0.27313430	0.6584046
4	4	healthy	0.4760534	-0.15884638	-0.27414570	0.7287284
5	5	healthy	0.3581253	-0.25861496	-0.41095167	0.4611839

```

6      6 healthy 0.4282631 -0.21881961 -0.35906808 0.6627319
7      7 healthy 0.2751988 -0.22427574 -0.36638109 0.5862114
8      8 healthy 0.4223983 -0.12988254 -0.22990450 0.7658009
9      9 healthy 0.4334381 -0.04661848 -0.08908399 0.6106513
10     10 healthy 0.4001629 -0.23359699 -0.37872496 0.7090615
      geometry
1 POINT (779838.5 2560800)
2 POINT (779901.2 2560885)
3 POINT (779920.1 2560971)
4 POINT (779837.5 2561010)
5 POINT (779595.2 2560779)
6 POINT (779584.6 2560739)
7 POINT (779569.6 2560857)
8 POINT (779729.4 2560865)
9 POINT (779710.5 2560803)
10 POINT (780078.9 2560879)

```

As we will not intend to use the geometry in our model, we can remove it:

```
st_geometry(nsamples) <- NULL
```

Let's check the output:

```
nsamples
```

	OBJECTID	score	ndvi	ndre	cire	sipi
1	1	healthy	0.39976674	-0.142773261	-0.24987155	0.7113871
2	2	healthy	0.47045520	-0.072608196	-0.13538624	0.7160328
3	3	healthy	0.42184072	-0.158167658	-0.27313430	0.6584046
4	4	healthy	0.47605337	-0.158846378	-0.27414570	0.7287284
5	5	healthy	0.35812534	-0.258614958	-0.41095167	0.4611839
6	6	healthy	0.42826313	-0.218819608	-0.35906808	0.6627319
7	7	healthy	0.27519883	-0.224275739	-0.36638109	0.5862114
8	8	healthy	0.42239829	-0.129882540	-0.22990450	0.7658009
9	9	healthy	0.43343808	-0.046618476	-0.08908399	0.6106513
10	10	healthy	0.40016295	-0.233596986	-0.37872496	0.7090615
11	11	healthy	0.39041685	-0.228953927	-0.37259969	0.6571411
12	12	healthy	0.42558618	-0.140911613	-0.24701583	0.5150259
13	13	healthy	0.49340855	-0.196911688	-0.32903294	0.6761255
14	14	wilted	0.05874133	-0.068648057	-0.12847645	3.4707223
15	15	wilted	0.38400155	0.038426164	0.07992348	0.7087015
16	16	wilted	0.20475192	0.099664192	0.22139338	1.1451788
17	17	wilted	0.23249338	-0.012808011	-0.02529208	0.9541926
18	18	wilted	0.07308271	0.059516988	0.12656685	3.3069715
19	19	wilted	0.09034886	0.050902211	0.10726442	-1.1224448
20	20	wilted	0.16015626	0.166594498	0.39979217	1.9539515
21	21	wilted	0.23390146	0.150849653	0.35529551	0.7991395
22	22	wilted	0.37398272	-0.026842737	-0.05228208	0.7405385
23	23	wilted	0.49255873	0.128742778	0.29553334	0.7991471
24	24	wilted	0.31349498	-0.271995471	-0.42766736	0.6524136
25	25	wilted	0.30539446	-0.170267124	-0.29098848	0.6172290
26	26	wilted	0.37186502	0.007804645	0.01573207	0.7467369
27	27	wilted	0.11435660	0.048839732	0.10269506	1.4091384
28	28	wilted	0.46834754	0.009134113	0.01843663	0.8078043
29	29	wilted	0.16656825	0.018023671	0.03670897	1.4190911
30	30	wilted	0.42070587	-0.144575443	-0.25262720	0.5272193
31	31	wilted	0.36820886	0.058621948	0.12454497	0.8245829

```

32      32 wilted 0.20858497 -0.056220022 -0.10645513 1.1397723
33      33 wilted 0.21998168 -0.119623392 -0.21368505 1.2761610
34      34 wilted 0.28577459 -0.015965114 -0.03142847 0.8868655
35      35 wilted 0.51481347 0.007888690 0.01590283 0.7890558
36      36 wilted 0.26034165 -0.186300667 -0.31408676 0.6313475
37      37 wilted 0.35448491 0.049444260 0.10403232 0.7842711
38      38 wilted 0.35485134 0.049169099 0.10342343 0.3232732
39      39 wilted 0.41155355 -0.075256088 -0.13997798 0.3632746
40      40 wilted 0.27764292 -0.043296886 -0.08300012 0.6920519
41      41 wilted 0.47575852 0.012897741 0.02613253 0.7788887
42      42 wilted 0.28374316 0.100019670 0.22227079 1.0312235
43      43 wilted 0.43536054 -0.005372302 -0.01068719 0.8065662
44      44 wilted 0.36866873 -0.023050553 -0.04506239 0.5711057
45      45 wilted 0.33280476 -0.058164786 -0.10993521 0.6822160
46      46 wilted 0.51897525 -0.080920969 -0.14972597 0.7637551
47      47 wilted 0.51320069 -0.029567154 -0.05743609 0.9153090
48      48 wilted 0.36543237 0.053204886 0.11238944 1.0314492
49      49 healthy 0.36090044 -0.279952871 -0.43744247 0.5027884
50      50 wilted 0.35309882 -0.001355745 -0.00270782 0.7946941
51      51 healthy 0.39490497 -0.130486894 -0.23085079 0.6122870
52      52 healthy 0.42757915 -0.240080432 -0.38720139 0.5241772
53      53 healthy 0.37856775 -0.233516246 -0.37861884 0.7662385
54      54 healthy 0.45668180 -0.197173474 -0.32939833 0.6839776
55      55 healthy 0.40846426 -0.148771702 -0.25901004 0.5934538
56      56 healthy 0.44160665 -0.192621610 -0.32302217 0.5988123
57      57 healthy 0.52035481 -0.141993366 -0.24867634 0.5994354
58      58 healthy 0.36400542 -0.238072742 -0.38458603 0.6079211
59      59 healthy 0.33408493 -0.228534207 -0.37204370 0.5059822
60      60 healthy 0.34397502 -0.200302031 -0.33375272 0.5625230
61      61 healthy 0.37706210 -0.348438178 -0.51680260 0.6534776
62      62 healthy 0.36836135 -0.066868925 -0.12535546 0.4413452
63      63 healthy 0.38415111 -0.180227754 -0.30541182 0.6743726
64      64 healthy 0.28650043 -0.116778741 -0.20913496 0.7196425
65      65 healthy 0.37453715 -0.251235648 -0.40158007 0.5122386
66      66 healthy 0.43549960 -0.237002851 -0.38318885 0.6752221
67      67 healthy 0.41255324 -0.191220292 -0.32104942 0.6745082
68      68 healthy 0.36656292 -0.238273304 -0.38484768 0.6054673
69      69 healthy 0.47892785 -0.215340798 -0.35437105 0.7113262
70      70 healthy 0.40082514 -0.279709014 -0.43714471 0.7465864
71      71 healthy 0.33697212 -0.181384162 -0.30707058 0.5021931
72      72 healthy 0.40791291 -0.222465226 -0.36396164 0.6068031
73      73 healthy 0.55198422 0.009196845 0.01856442 0.8259194
74      74 healthy 0.27583200 -0.288149676 -0.44738540 0.5335927
75      75 healthy 0.46411580 -0.149982883 -0.26084368 0.7550614
76      76 healthy 0.41437170 -0.249396644 -0.39922733 0.6959438
77      77 wilted 0.42071386 -0.014630713 -0.02883948 0.7815805
78      78 wilted 0.44323734 0.035346511 0.07328333 0.8931209
79      79 wilted 0.42093366 0.097763989 0.21671489 0.8684168
80      80 wilted 0.26383824 0.045756804 0.09590177 1.0954009

```

As our task is a binary classification (i.e any site can be either healthy or wilted), the variable to estimate is a *factor* (not a character).

Let's change the data type of such variable:

```
nsamples$score = as.factor(nsamples$score)
```

Let's check the result:

nsamples

	OBJECTID	score	ndvi	ndre	cire	sipi
1	1	healthy	0.39976674	-0.142773261	-0.24987155	0.7113871
2	2	healthy	0.47045520	-0.072608196	-0.13538624	0.7160328
3	3	healthy	0.42184072	-0.158167658	-0.27313430	0.6584046
4	4	healthy	0.47605337	-0.158846378	-0.27414570	0.7287284
5	5	healthy	0.35812534	-0.258614958	-0.41095167	0.4611839
6	6	healthy	0.42826313	-0.218819608	-0.35906808	0.6627319
7	7	healthy	0.27519883	-0.224275739	-0.36638109	0.5862114
8	8	healthy	0.42239829	-0.129882540	-0.22990450	0.7658009
9	9	healthy	0.43343808	-0.046618476	-0.08908399	0.6106513
10	10	healthy	0.40016295	-0.233596986	-0.37872496	0.7090615
11	11	healthy	0.39041685	-0.228953927	-0.37259969	0.6571411
12	12	healthy	0.42558618	-0.140911613	-0.24701583	0.5150259
13	13	healthy	0.49340855	-0.196911688	-0.32903294	0.6761255
14	14	wilted	0.05874133	-0.068648057	-0.12847645	3.4707223
15	15	wilted	0.38400155	0.038426164	0.07992348	0.7087015
16	16	wilted	0.20475192	0.099664192	0.22139338	1.1451788
17	17	wilted	0.23249338	-0.012808011	-0.02529208	0.9541926
18	18	wilted	0.07308271	0.059516988	0.12656685	3.3069715
19	19	wilted	0.09034886	0.050902211	0.10726442	-1.1224448
20	20	wilted	0.16015626	0.166594498	0.39979217	1.9539515
21	21	wilted	0.23390146	0.150849653	0.35529551	0.7991395
22	22	wilted	0.37398272	-0.026842737	-0.05228208	0.7405385
23	23	wilted	0.49255873	0.128742778	0.29553334	0.7991471
24	24	wilted	0.31349498	-0.271995471	-0.42766736	0.6524136
25	25	wilted	0.30539446	-0.170267124	-0.29098848	0.6172290
26	26	wilted	0.37186502	0.007804645	0.01573207	0.7467369
27	27	wilted	0.11435660	0.048839732	0.10269506	1.4091384
28	28	wilted	0.46834754	0.009134113	0.01843663	0.8078043
29	29	wilted	0.16656825	0.018023671	0.03670897	1.4190911
30	30	wilted	0.42070587	-0.144575443	-0.25262720	0.5272193
31	31	wilted	0.36820886	0.058621948	0.12454497	0.8245829
32	32	wilted	0.20858497	-0.056220022	-0.10645513	1.1397723
33	33	wilted	0.21998168	-0.119623392	-0.21368505	1.2761610
34	34	wilted	0.28577459	-0.015965114	-0.03142847	0.8868655
35	35	wilted	0.51481347	0.007888690	0.01590283	0.7890558
36	36	wilted	0.26034165	-0.186300667	-0.31408676	0.6313475
37	37	wilted	0.35448491	0.049444260	0.10403232	0.7842711
38	38	wilted	0.35485134	0.049169099	0.10342343	0.3232732
39	39	wilted	0.41155355	-0.075256088	-0.13997798	0.3632746
40	40	wilted	0.27764292	-0.043296886	-0.08300012	0.6920519
41	41	wilted	0.47575852	0.012897741	0.02613253	0.7788887
42	42	wilted	0.28374316	0.100019670	0.22227079	1.0312235
43	43	wilted	0.43536054	-0.005372302	-0.01068719	0.8065662
44	44	wilted	0.36866873	-0.023050553	-0.04506239	0.5711057
45	45	wilted	0.33280476	-0.058164786	-0.10993521	0.6822160
46	46	wilted	0.51897525	-0.080920969	-0.14972597	0.7637551
47	47	wilted	0.51320069	-0.029567154	-0.05743609	0.9153090
48	48	wilted	0.36543237	0.053204886	0.11238944	1.0314492
49	49	healthy	0.36090044	-0.279952871	-0.43744247	0.5027884
50	50	wilted	0.35309882	-0.001355745	-0.00270782	0.7946941
51	51	healthy	0.39490497	-0.130486894	-0.23085079	0.6122870
52	52	healthy	0.42757915	-0.240080432	-0.38720139	0.5241772
53	53	healthy	0.37856775	-0.233516246	-0.37861884	0.7662385
54	54	healthy	0.45668180	-0.197173474	-0.32939833	0.6839776

```

55      55 healthy 0.40846426 -0.148771702 -0.25901004 0.5934538
56      56 healthy 0.44160665 -0.192621610 -0.32302217 0.5988123
57      57 healthy 0.52035481 -0.141993366 -0.24867634 0.5994354
58      58 healthy 0.36400542 -0.238072742 -0.38458603 0.6079211
59      59 healthy 0.33408493 -0.228534207 -0.37204370 0.5059822
60      60 healthy 0.34397502 -0.200302031 -0.33375272 0.5625230
61      61 healthy 0.37706210 -0.348438178 -0.51680260 0.6534776
62      62 healthy 0.36836135 -0.066868925 -0.12535546 0.4413452
63      63 healthy 0.38415111 -0.180227754 -0.30541182 0.6743726
64      64 healthy 0.28650043 -0.116778741 -0.20913496 0.7196425
65      65 healthy 0.37453715 -0.251235648 -0.40158007 0.5122386
66      66 healthy 0.43549960 -0.237002851 -0.38318885 0.6752221
67      67 healthy 0.41255324 -0.191220292 -0.32104942 0.6745082
68      68 healthy 0.36656292 -0.238273304 -0.38484768 0.6054673
69      69 healthy 0.47892785 -0.215340798 -0.35437105 0.7113262
70      70 healthy 0.40082514 -0.279709014 -0.43714471 0.7465864
71      71 healthy 0.33697212 -0.181384162 -0.30707058 0.5021931
72      72 healthy 0.40791291 -0.222465226 -0.36396164 0.6068031
73      73 healthy 0.55198422 0.009196845 0.01856442 0.8259194
74      74 healthy 0.27583200 -0.288149676 -0.44738540 0.5335927
75      75 healthy 0.46411580 -0.149982883 -0.26084368 0.7550614
76      76 healthy 0.41437170 -0.249396644 -0.39922733 0.6959438
77      77 wilted 0.42071386 -0.014630713 -0.02883948 0.7815805
78      78 wilted 0.44323734 0.035346511 0.07328333 0.8931209
79      79 wilted 0.42093366 0.097763989 0.21671489 0.8684168
80      80 wilted 0.26383824 0.045756804 0.09590177 1.0954009

```

A simple summary of the extracted data can be useful:

```

nsamples |>
  group_by(score) |>
  summarize(n())

```

```

# A tibble: 2 × 2
  score   `n()`
  <fct>   <int>
1 healthy    40
2 wilted    40

```

This means the dataset is balanced which is very good.

Saving the extracted dataset

Now, let's save the *nsamples* object. Just in case R crashes due to lack of memory.

```

#uncomment if needed
#st_write(nsamples, "./banana_data/nsamples.csv", overwrite=TRUE)

```

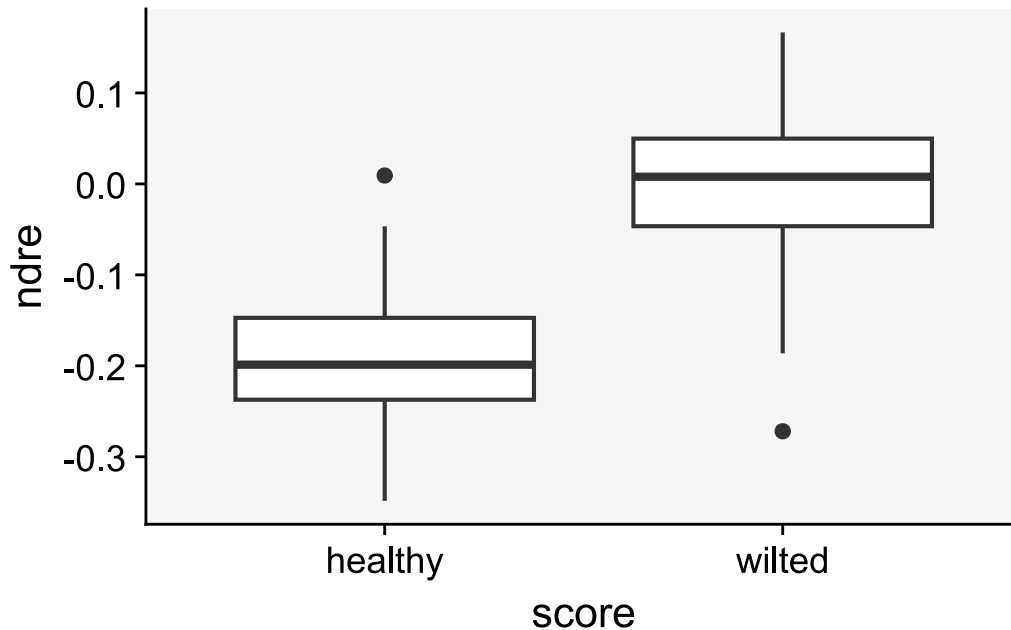
Classification of Fusarium wilt using machine learning (ML)

The overall process to classify the crop disease under study will be conducted using the *tidymodels* framework which is an extension of the *tidyverse* suite. It is especially focused towards providing a generalized way to define, run and optimize ML models in R.

Exploratory analysis

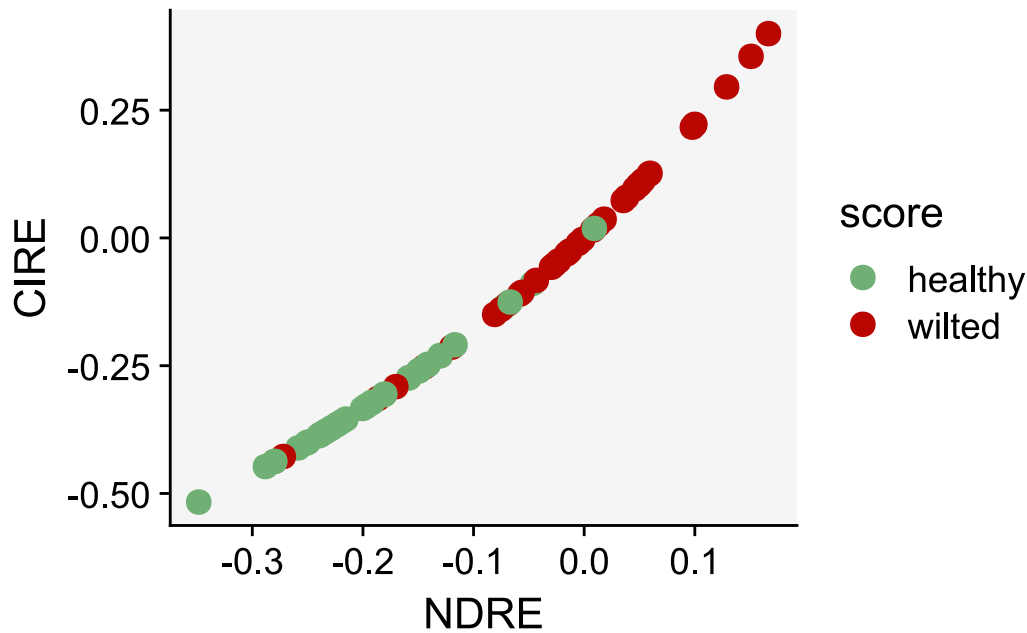
As a first step in modeling, it's always a good idea to visualize the data. Let's start with a *boxplot* to display the distribution of a vegetation index. It visualizes five summary statistics (the median, two hinges and two whiskers), and all "outlying" points individually.

```
p <- ggplot(nsamples, aes(score, ndre))+
  r4pde::theme_r4pde()
p + geom_boxplot()
```



Next, we will do a scatterplot to visualize the indices NDRE and CIRE:

```
ggplot(nsamples) +
  aes(x = ndre, y = cire, color = score) +
  geom_point(shape = 16, size = 4) +
  labs(x = "NDRE", y = "CIRE") +
  r4pde::theme_r4pde() +
  scale_color_manual(values = c("#71b075", "#ba0600"))
```



Splitting the data

Next step is to divide the data into a training and a test set. The `set.seed()` function can be used for reproducibility of the computations that are dependent on random numbers. By default, the training/testing split is 0.75 to 0.25.

```
set.seed(42)
data_split <- initial_split(data = nsamples)
data_train <- training(data_split)
data_test <- testing(data_split)
```

Let's check the result:

```
data_train
```

	OBJECTID	score	ndvi	ndre	cire	sipi
1	49	healthy	0.36090044	-0.279952871	-0.43744247	0.5027884
2	65	healthy	0.37453715	-0.251235648	-0.40158007	0.5122386
3	25	wilted	0.30539446	-0.170267124	-0.29098848	0.6172290
4	74	healthy	0.27583200	-0.288149676	-0.44738540	0.5335927
5	18	wilted	0.07308271	0.059516988	0.12656685	3.3069715
6	80	wilted	0.26383824	0.045756804	0.09590177	1.0954009
7	47	wilted	0.51320069	-0.029567154	-0.05743609	0.9153090
8	24	wilted	0.31349498	-0.271995471	-0.42766736	0.6524136
9	71	healthy	0.33697212	-0.181384162	-0.30707058	0.5021931
10	37	wilted	0.35448491	0.049444260	0.10403232	0.7842711
11	20	wilted	0.16015626	0.166594498	0.39979217	1.9539515
12	26	wilted	0.37186502	0.007804645	0.01573207	0.7467369
13	3	healthy	0.42184072	-0.158167658	-0.27313430	0.6584046
14	41	wilted	0.47575852	0.012897741	0.02613253	0.7788887
15	27	wilted	0.11435660	0.048839732	0.10269506	1.4091384
16	36	wilted	0.26034165	-0.186300667	-0.31408676	0.6313475
17	72	healthy	0.40791291	-0.222465226	-0.36396164	0.6068031
18	31	wilted	0.36820886	0.058621948	0.12454497	0.8245829
19	45	wilted	0.33280476	-0.058164786	-0.10993521	0.6822160
20	5	healthy	0.35812534	-0.258614958	-0.41095167	0.4611839

21	70	healthy	0.40082514	-0.279709014	-0.43714471	0.7465864
22	34	wilted	0.28577459	-0.015965114	-0.03142847	0.8868655
23	28	wilted	0.46834754	0.009134113	0.01843663	0.8078043
24	40	wilted	0.27764292	-0.043296886	-0.08300012	0.6920519
25	68	healthy	0.36656292	-0.238273304	-0.38484768	0.6054673
26	33	wilted	0.21998168	-0.119623392	-0.21368505	1.2761610
27	42	wilted	0.28374316	0.100019670	0.22227079	1.0312235
28	73	healthy	0.55198422	0.009196845	0.01856442	0.8259194
29	30	wilted	0.42070587	-0.144575443	-0.25262720	0.5272193
30	43	wilted	0.43536054	-0.005372302	-0.01068719	0.8065662
31	15	wilted	0.38400155	0.038426164	0.07992348	0.7087015
32	22	wilted	0.37398272	-0.026842737	-0.05228208	0.7405385
33	8	healthy	0.42239829	-0.129882540	-0.22990450	0.7658009
34	79	wilted	0.42093366	0.097763989	0.21671489	0.8684168
35	4	healthy	0.47605337	-0.158846378	-0.27414570	0.7287284
36	75	healthy	0.46411580	-0.149982883	-0.26084368	0.7550614
37	76	healthy	0.41437170	-0.249396644	-0.39922733	0.6959438
38	58	healthy	0.36400542	-0.238072742	-0.38458603	0.6079211
39	61	healthy	0.37706210	-0.348438178	-0.51680260	0.6534776
40	46	wilted	0.51897525	-0.080920969	-0.14972597	0.7637551
41	59	healthy	0.33408493	-0.228534207	-0.37204370	0.5059822
42	35	wilted	0.51481347	0.007888690	0.01590283	0.7890558
43	53	healthy	0.37856775	-0.233516246	-0.37861884	0.7662385
44	23	wilted	0.49255873	0.128742778	0.29553334	0.7991471
45	69	healthy	0.47892785	-0.215340798	-0.35437105	0.7113262
46	6	healthy	0.42826313	-0.218819608	-0.35906808	0.6627319
47	39	wilted	0.41155355	-0.075256088	-0.13997798	0.3632746
48	2	healthy	0.47045520	-0.072608196	-0.13538624	0.7160328
49	60	healthy	0.34397502	-0.200302031	-0.33375272	0.5625230
50	56	healthy	0.44160665	-0.192621610	-0.32302217	0.5988123
51	62	healthy	0.36836135	-0.066868925	-0.12535546	0.4413452
52	21	wilted	0.23390146	0.150849653	0.35529551	0.7991395
53	55	healthy	0.40846426	-0.148771702	-0.25901004	0.5934538
54	64	healthy	0.28650043	-0.116778741	-0.20913496	0.7196425
55	57	healthy	0.52035481	-0.141993366	-0.24867634	0.5994354
56	10	healthy	0.40016295	-0.233596986	-0.37872496	0.7090615
57	48	wilted	0.36543237	0.053204886	0.11238944	1.0314492
58	54	healthy	0.45668180	-0.197173474	-0.32939833	0.6839776
59	1	healthy	0.39976674	-0.142773261	-0.24987155	0.7113871
60	17	wilted	0.23249338	-0.012808011	-0.02529208	0.9541926

Defining the model

We will use a logistic regression which is a simple model. It may be useful to have a look at [this explanation](#) of such a model.

```
spec_lr <-
  logistic_reg() |>
  set_engine("glm") |>
  set_mode("classification")
```

Defining the recipe

The `recipe()` function to be used here has two arguments:

- A formula. Any variable on the left-hand side of the tilde (~) is considered the model outcome (here, outcome). On the right-hand side of the tilde are the predictors. Variables may be listed by name, or you can use the dot (.) to indicate all other variables as predictors.

- The data. A recipe is associated with the data set used to create the model. This will typically be the training set, so `data = data_train` here.

```
recipe_lr <-
  recipe(score ~ ., data_train) |>
  add_role(OBJECTID, new_role = "id") |>
  step_zv(all_predictors()) |>
  step_corr(all_predictors())
```

Evaluating model performance

Next, we need to specify what we would like to see for determining the performance of the model. Different modelling algorithms have different types of metrics. Because we have a binary classification problem (healthy vs. wilted classification), we will choose the AUC - ROC evaluation metric here.

Combining model and recipe into a workflow

We will want to use our recipe across several steps as we train and test our model. We will:

- Process the recipe using the training set: This involves any estimation or calculations based on the training set. For our recipe, the training set will be used to determine which predictors should be converted to dummy variables and which predictors will have zero-variance in the training set, and should be slated for removal.
- Apply the recipe to the training set: We create the final predictor set on the training set.
- Apply the recipe to the test set: We create the final predictor set on the test set. Nothing is recomputed and no information from the test set is used here; the dummy variable and zero-variance results from the training set are applied to the test set.

To simplify this process, we can use a model workflow, which pairs a model and recipe together. This is a straightforward approach because different recipes are often needed for different models, so when a model and recipe are bundled, it becomes easier to train and test workflows. We'll use the workflows package from *tidymodels* to bundle our model with our recipe.

Now we are ready to set up our complete modelling workflow. This workflow contains the model specification and the recipe.

```
wf_bana_wilt <-
  workflow(
    spec = spec_lr,
    recipe_lr
  )

wf_bana_wilt
```

```
== Workflow ==
Preprocessor: Recipe
Model: logistic_reg()

— Preprocessor —
2 Recipe Steps

• step_zv()
• step_corr()
```

```
— Model —
Logistic Regression Model Specification (classification)

Computational engine: glm
```

Fitting the logistic regression model

Now we use the workflow previously created to fit the model on our training data. We use the training partition of the data.

```
fit_lr <- wf_bana_wilt |>
  fit(data = data_train)
```

Let's check the output:

```
fit_lr
```

```
== Workflow [trained] ==
Preprocessor: Recipe
Model: logistic_reg()

— Preprocessor —
2 Recipe Steps

• step_zv()
• step_corr()

— Model —

Call: stats::glm(formula = ..y ~ ., family = stats::binomial, data = data)

Coefficients:
(Intercept)    OBJECTID         ndvi         ndre         sipi
    9.27889    -0.02585   -16.62083    24.61264     1.51054

Degrees of Freedom: 59 Total (i.e. Null); 55 Residual
Null Deviance:      83.18
Residual Deviance: 31.96   AIC: 41.96
```

Now, we will use the fitted model to estimate health status in the training data:

```
rf_training_pred <-
  predict(fit_lr, data_train) |>
  bind_cols(predict(fit_lr, data_train, type = "prob")) |>
  # Add the true outcome data back in
  bind_cols(data_train |>
    select(score))
```

What we got?

```
rf_training_pred
```

```
# A tibble: 60 × 4
  .pred_class .pred_healthy .pred_wilted score
  <fct>       <dbl>         <dbl> <fct>
1 healthy     0.984         0.0160 healthy
```

```

2 healthy      0.983      0.0174 healthy
3 wilted       0.426      0.574  wilted
4 healthy      0.971      0.0292 healthy
5 wilted       0.000000784  1.000  wilted
6 wilted       0.00366      0.996  wilted
7 wilted       0.453      0.547  wilted
8 healthy      0.906      0.0944 wilted
9 healthy      0.866      0.134  healthy
10 wilted      0.00791      0.992  wilted
# i 50 more rows

```

Let's estimate the training accuracy:

```

rf_training_pred |> # training set predictions
  accuracy(truth = score, .pred_class) -> acc_train
acc_train

```

```

# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>    <chr>         <dbl>
1 accuracy binary      0.867

```

The accuracy of the model on the training data is 0.87, which is above 0.5 (the accuracy expected by chance for two balanced classes). This basically means that the model was able to learn predictive patterns from the training data. To see if the model is able to generalize what it learned when exposed to new data, we evaluate the model on our hold-out (or so-called test data). We created a test dataset when splitting the data at the start of the modelling.

Evaluating the model on test data

Now, we will use the fitted model to estimate health status in the testing data:

```

lr_testing_pred <-
  predict(fit_lr, data_test) |>
  bind_cols(predict(fit_lr, data_test, type = "prob")) |>
  bind_cols(data_test |> select(score))

```

What we got:

```
lr_testing_pred
```

```

# A tibble: 20 × 4
  .pred_class .pred_healthy .pred_wilted score
  <fct>       <dbl>         <dbl> <fct>
1 healthy     0.528         0.472 healthy
2 wilted      0.166         0.834 healthy
3 healthy     0.894         0.106 healthy
4 healthy     0.689         0.311 healthy
5 healthy     0.956         0.0438 healthy
6 wilted      0.0000102    1.000  wilted
7 wilted      0.0000647    1.000  wilted
8 wilted      0.00107         0.999  wilted
9 wilted      0.000237     1.000  wilted
10 wilted     0.00486         0.995  wilted
11 wilted     0.0163         0.984  wilted
12 wilted     0.0904         0.910  wilted

```


13 wilted	0.0361	0.964 wilted
14 healthy	0.709	0.291 healthy
15 healthy	0.986	0.0135 healthy
16 healthy	0.896	0.104 healthy
17 healthy	0.989	0.0112 healthy
18 healthy	0.952	0.0475 healthy
19 wilted	0.247	0.753 wilted
20 wilted	0.108	0.892 wilted

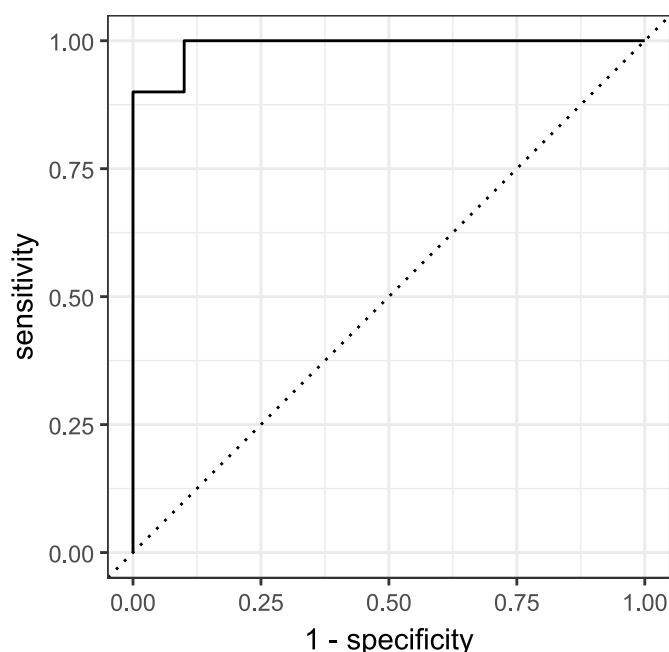
Let's compute the testing accuracy:

```
lr_testing_pred |> # test set predictions
  accuracy(score, .pred_class)
```

```
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>       <dbl>
1 accuracy binary      0.95
```

The accuracy on the test data is similar to or higher than the accuracy on the training data, so there is no sign of overfitting. Note that the test set has only 20 samples, so this estimate is imprecise. The result is good for a first go and a relatively simple classification model.

```
## Let's plot the AUC-ROC
lr_testing_pred |>
  roc_curve(truth = score, .pred_wilted, event_level="second") |>
  mutate(model = "Logistic Regression") |>
  autoplot()
```



Conclusions

In this section, we trained and tested a logistic regression model (LGM) using four spectral indices as predictor variables (i.e. NDVI, NDRE, CIRE and SIPI). Compare this section results, in terms of equation and accuracy, with the individual LGMs tested by [Ye et al. \(2020\)](#).

Table 3. The logistic regression models for different vegetation indices.

VI	Logistic Regression Equation	OA* of the Fitting (%)	Validation Dataset 1		Validation Dataset 2	
			OA (%)	Kappa	OA (%)	Kappa
NDVI	$y = -11.851 \times \text{NDVI} + 5.373$	86.3	83.3	0.66	62.9	0.22
NDRE	$y = -15.775 \times \text{NDRE} + 1.802$	90.5	87.5	0.75	65.7	0.39
CI _{green}	$y = -4.144 \times \text{CI}_{\text{green}} + 3.118$	89.5	87.5	0.74	74.3	0.47
CI _{RE}	$y = -6.110 \times \text{CI}_{\text{RE}} + 1.935$	91.6	91.7	0.83	80.0	0.59
CARI	$y = -9.966 \times \text{CARI} + 3.172$	62.1	66.7	0.35	60.0	0.21
ARI	$y = -7.247 \times \text{ARI} + 5.326$	75.8	83.3	0.66	68.6	0.37

* Overall accuracy.

Figure 8.12

Note that we have not tested other ML algorithms. But there are a lot of them available from the *tidymodels* framework (e.g. random forests, support vector machines, gradient boosting machines).

To conclude, this section illustrated how to use VIs derived from UAV-based multispectral imagery and ground data to develop an identification model for detecting banana Fusarium wilt. The results showed that a simple logistic regression model is able to identify Fusarium wilt of banana from several VIs with a good accuracy. However, before going too optimistic, I would suggest to study the [Ye et al. \(2020\)](#) paper and critically evaluate their experiment design, methods and results.

Disease quantification

Introduction

This section illustrates the use of UAV multispectral imagery for estimating the severity of cercospora leaf spot (CLS) disease in table beet, the most destructive fungal disease of table beet ([Skaracis et al., 2010](#); [Tan et al., 2023](#)). The CLS disease causes rapid defoliation and significant crop loss may occur through the inability to harvest with top-pulling machinery. It may also render the produce unsaleable for the fresh market ([Skaracis et al., 2010](#)).

[Saif et al. \(2024\)](#) recently published on [Mendeley data](#) UAV imagery & ground truth CLS data collected at several table beet plots at Cornell AgriTech, Geneva, New York, USA. Note that, in this study, UAV multispectral images were collected using a Micasense Red Edge camera, similar to the one used in the case study for the detection of Fusarium wilt on banana. I have not found any scientific paper estimating CLS severity from this dataset. However, in [Saif et al. \(2023\)](#), a similar dataset was used to forecast table beet root yield at Cornell Agritech. While it is just a guess, it is worth visualizing the plots in the latter study.

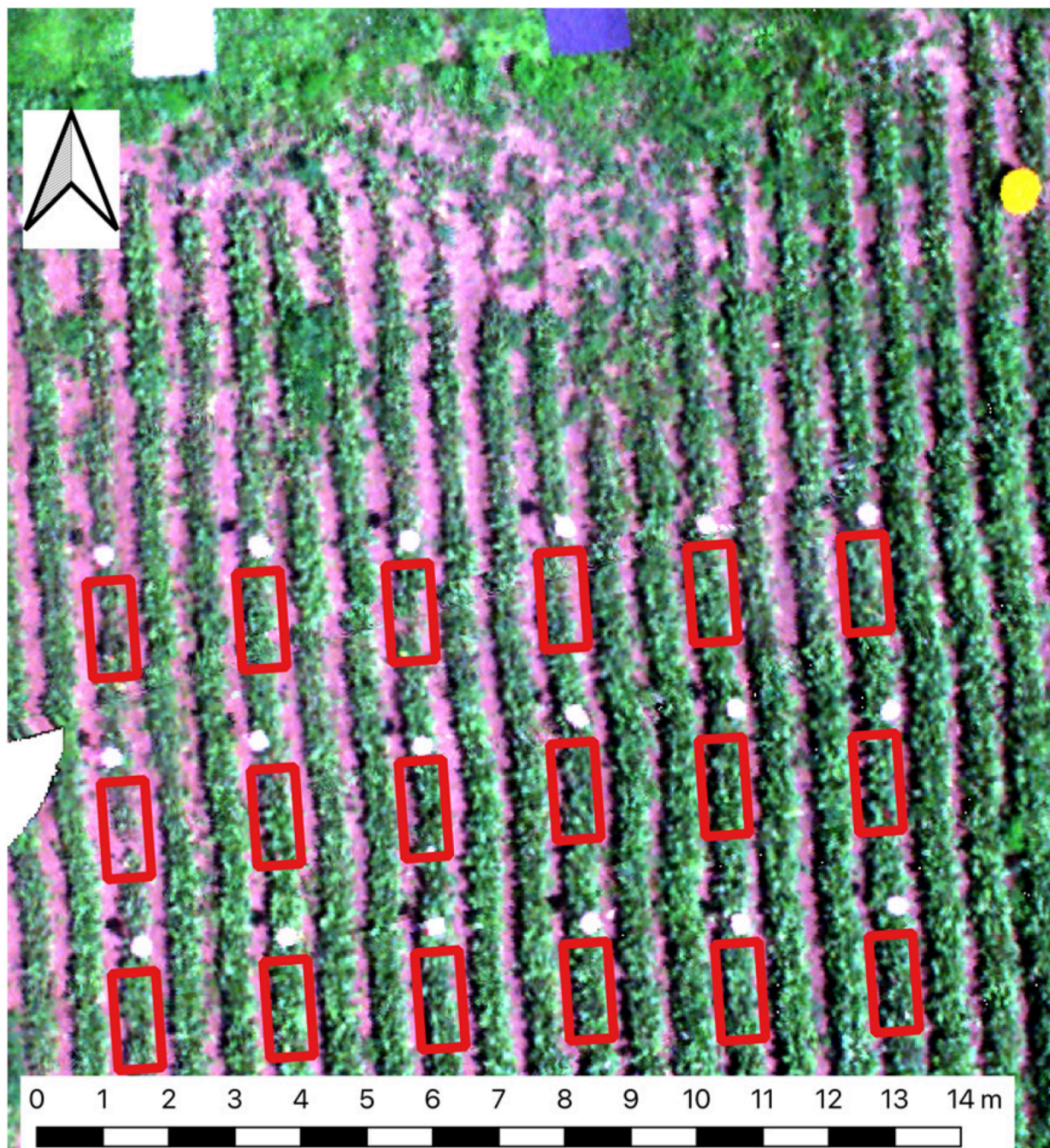


Figure 8.13: Color mosaic of third flight hyperspectral image. The visualization bands are the following: red (639 nm), green (550 nm), and blue (470 nm). The crops in the red boxes in the figure were the plots under study. Source: (Saif et al., 2023)

This section aims at estimating CLS leaf severity using vegetation indices (VIs) derived from the multispectral UAV imagery as spectral covariates. The section comprises two parts: (i) Part 1 creates a spectral covariate table to summarize the complete UAV imagery & ground truth data; and (ii) Part 2 trains and tests a machine learning model to estimate CLS severity.

CLS severity on leaves

It is very important to visualize how different levels of table beet CLS disease severity look in RGB color leaf images. The figure below is a standard area diagram set (SADs) that is used by

raters during the assessment of visual severity to increase their accuracy and reliability of the estimates.

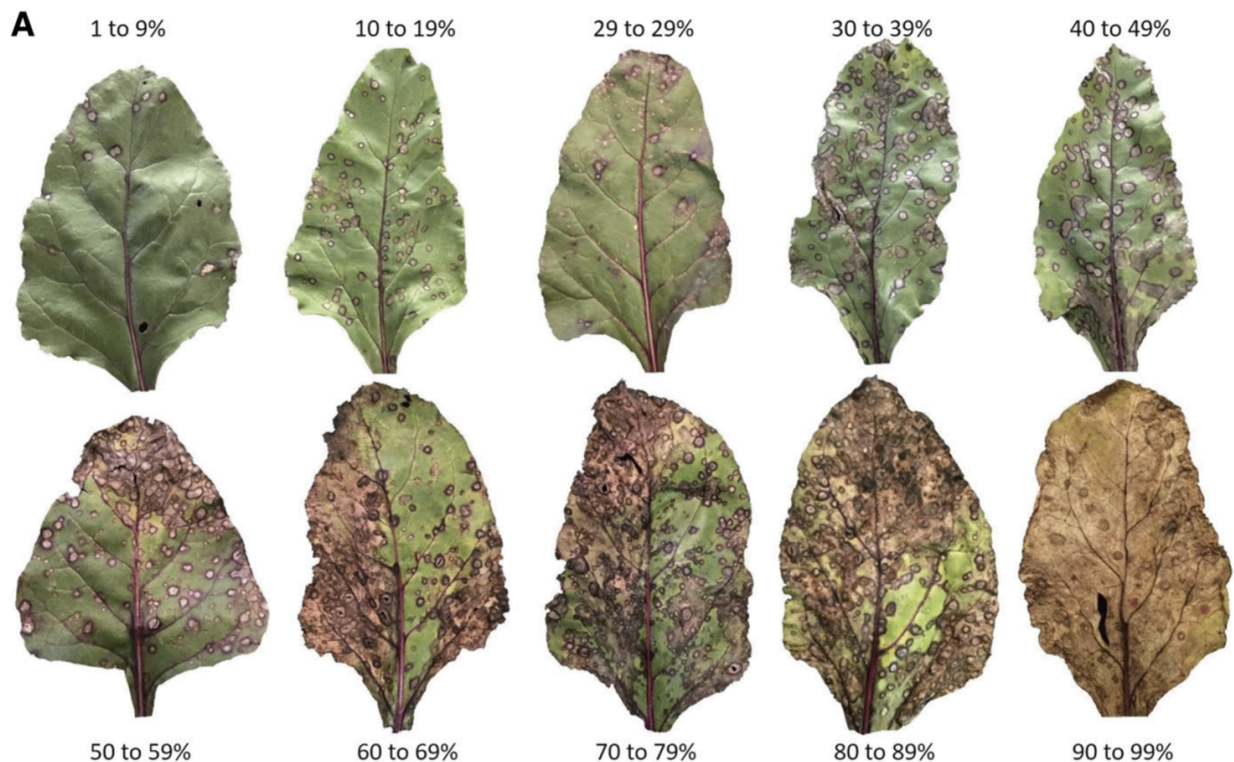


Figure 8.14: Standard area diagram set for the severity of table beet cercospora leaf spot. Source: (Del Ponte et al., 2019)

Software setup

Let's start by cleaning up R memory:

```
# Re-enable execution for the Cercospora leaf spot section
knitr::opts_chunk$set(eval = TRUE)
rm(list=ls())
```

Then, we need to install several packages (if they are not installed yet):

```
list.of.packages <- c("readr", "terra", "tidyterra", "stars", "sf", "leaflet", "leafem", "dplyr", "ggplot2",
"tidymodels")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)
```

Now, let's load all the required packages:

```
library(readr)
library(terra)
library(tidyterra)
library(stars)
library(sf)
library(leaflet)
library(leafem)
library(dplyr)
```

```
library(ggplot2)
library(tidymodels)
```

Read the dataset

The following code assumes you have already downloaded the `dataset` and unzipped its content under the `data/cercospora_data` directory. What files are in that folder?

```
list.files("data/cercospora_data")
```

```
[1] "CLS_DS.csv"          "D0_2021.csv"         "D1_2021.csv"
[4] "D2_2021.csv"         "D3_2021.csv"         "D4_2021.csv"
[7] "fcovar_2021.csv"     "multispec_2021_2022" "multispec_2023"
[10] "ncovar_2021.csv"     "README.md"
```

Note that there is one CLS_DS.csv file with the CLS ground data and two folders with the UAV multispectral images. We can infer that 2021, 2022, 2023 refer to the image acquisition years. At this point, it is very important to check the README file. I will summarize the following points:

File naming convention

Each image file is named according to the plot number and the date of capture, using the format `plt_rYYYYMMDD`, where:

- **'plt'** stands for the plot number.
- **'YYYYMMDD'** represents the date on which the image was captured (year, month, day).

For example, the file name `5_r20210715` corresponds to an image taken on July 15, 2021, from study plot 5.

CLS Severity Data

The file named CLS_DS contains visual assessments of CLS disease severity noted for each plot.

Inspect the format of each file

Let's list the first images under an image folder:

```
list.files("data/cercospora_data/multispec_2021_2022/")[1:15]
```

```
[1] "1_r20210707.tif"          "1_r20210707.tif.aux.xml"
[3] "1_r20210715.tif"          "1_r20210720.tif"
[5] "1_r20210802.tif"          "1_r20210825.tif"
[7] "1_r20210825.tif.aux.xml"  "1_r20220707.tif"
[9] "1_r20220715.tif"          "1_r20220726.tif"
[11] "1_r20220810.tif"          "1_r20220818.tif"
[13] "10_r20210707.tif"         "10_r20210715.tif"
[15] "10_r20210720.tif"
```

Note that, for 2021, there are six dates of image acquisition: 20210707, 20210715, 20210720, 20210726, 20210802, 20210825

Read the orthomosaics

Now, let's read several plot images using the *terra* package:

```
# Open a tif collected on study plot 5 on 7th July 2021
tif1 <- "data/cercospora_data/multispec_2021_2022/1_r20210707.tif"
tif2 <- "data/cercospora_data/multispec_2021_2022/2_r20210707.tif"
tif3 <- "data/cercospora_data/multispec_2021_2022/3_r20210707.tif"
tif4 <- "data/cercospora_data/multispec_2021_2022/4_r20210707.tif"
tif5 <- "data/cercospora_data/multispec_2021_2022/5_r20210707.tif"
tif6 <- "data/cercospora_data/multispec_2021_2022/6_r20210707.tif"
tif7 <- "data/cercospora_data/multispec_2021_2022/7_r20210707.tif"
tif8 <- "data/cercospora_data/multispec_2021_2022/8_r20210707.tif"
tif9 <- "data/cercospora_data/multispec_2021_2022/9_r20210707.tif"
tif10 <- "data/cercospora_data/multispec_2021_2022/10_r20210707.tif"
```

It may be convenient to increase image pixel size (to make the raster lighter):

```
p01 <- terra::rast(tif1) |> aggregate(20)
p02 <- terra::rast(tif2) |> aggregate(20)
p03 <- terra::rast(tif3) |> aggregate(20)
p04 <- terra::rast(tif4) |> aggregate(20)
p05 <- terra::rast(tif5) |> aggregate(20)
p06 <- terra::rast(tif6) |> aggregate(20)
p07 <- terra::rast(tif7) |> aggregate(20)
p08 <- terra::rast(tif8) |> aggregate(20)
p09 <- terra::rast(tif9) |> aggregate(20)
p10 <- terra::rast(tif10) |> aggregate(20)
```

Let's check what we get:

```
p01
```

```
class      : SpatRaster
size       : 15, 9, 5  (nrow, ncol, nlyr)
resolution : 0.212, 0.212  (x, y)
extent     : 334196.9, 334198.8, 4748990, 4748993  (xmin, xmax, ymin, ymax)
coord. ref.: WGS 84 / UTM zone 18N (EPSG:32618)
source(s)  : memory
names      : 1_r20210707_1, 1_r20210707_2, 1_r20210707_3, 1_r20210707_4, 1_r20210707_5
min values : 0.01288329, 0.02743819, 0.01721323, 0.06647335, 0.1167282
max values : 0.03016389, 0.06923942, 0.07651012, 0.18791453, 0.4155874
```

Note that each image has 5 bands with ~21 cm pixel size.

As the images have been split in plots, it may be useful to "mosaic" them.

```
# with many SpatRasters, make a SpatRasterCollection from a list
rlist <- list(p01, p02, p03, p04, p05, p06, p07, p08, p09, p10)
rsrc <- sprc(rlist)

m <- merge(rsrc)
```

```
Warning: [merge] 10 raster(s) that did not share the base geometry of the first
raster were resampled
```

What we got?

```
m
```

```

class      : SpatRaster
size       : 209, 24, 5  (nrow, ncol, nlyr)
resolution : 0.212, 0.212  (x, y)
extent     : 334193.8, 334198.9, 4748990, 4749035  (xmin, xmax, ymin, ymax)
coord. ref.: WGS 84 / UTM zone 18N (EPSG:32618)
source(s)  : memory
names      : 1_r20210707_1, 1_r20210707_2, 1_r20210707_3, 1_r20210707_4, 1_r20210707_5
min values : 0.01390121, 0.02954650, 0.01960831, 0.0706145, 0.1283448
max values : 0.03741143, 0.08793678, 0.08386827, 0.2208381, 0.5030925

```

Let's check the band names:

```
names(m)
```

```

[1] "1_r20210707_1" "1_r20210707_2" "1_r20210707_3" "1_r20210707_4"
[5] "1_r20210707_5"

```

We will rename band names:

```

# Rename
m2 <- m |>
  rename(blue = "1_r20210707_1", green = "1_r20210707_2",
         red = "1_r20210707_3", redge = "1_r20210707_4",
         nir = "1_r20210707_5")

```

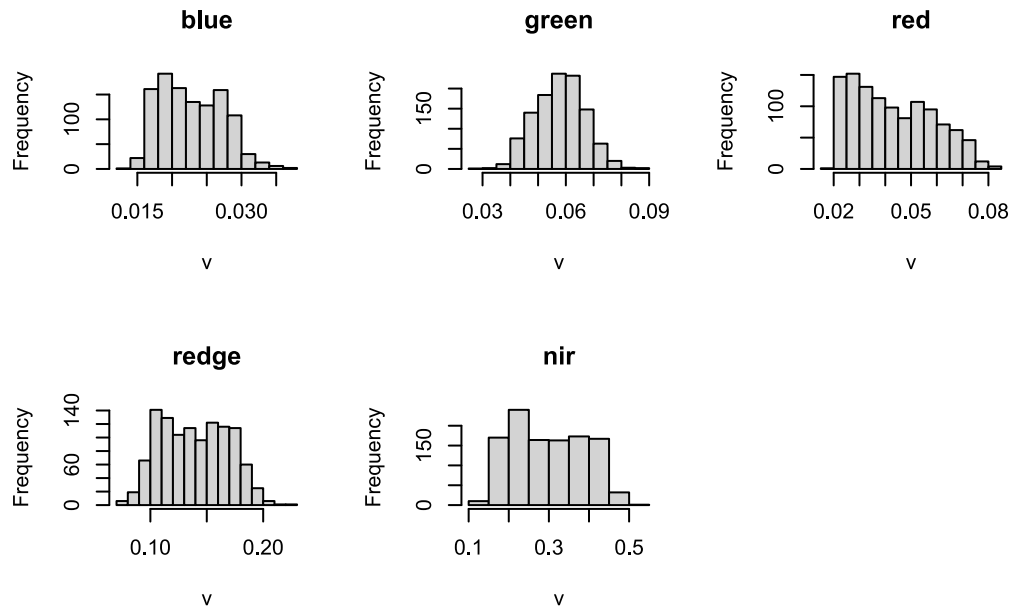
Let's get a summary of the mosaic:

```
summary(m2)
```

blue	green	red	redge
Min. :0.0139	Min. :0.0295	Min. :0.0196	Min. :0.0706
1st Qu.:0.0189	1st Qu.:0.0516	1st Qu.:0.0291	1st Qu.:0.1133
Median :0.0222	Median :0.0576	Median :0.0410	Median :0.1378
Mean :0.0228	Mean :0.0578	Mean :0.0431	Mean :0.1391
3rd Qu.:0.0265	3rd Qu.:0.0639	3rd Qu.:0.0556	3rd Qu.:0.1645
Max. :0.0374	Max. :0.0879	Max. :0.0839	Max. :0.2208
NA's :3896	NA's :3896	NA's :3896	NA's :3896

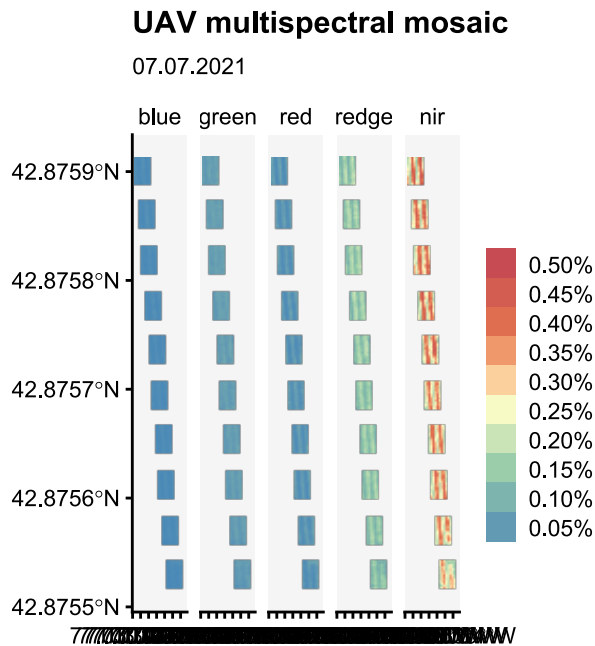
nir
Min. :0.1283
1st Qu.:0.2162
Median :0.2912
Mean :0.2985
3rd Qu.:0.3767
Max. :0.5031
NA's :3896

```
hist(m2)
```



Let's visualize the mosaic:

```
ggplot() +
  geom_spatraster(data = m2) +
  facet_wrap(~lyr, ncol = 5) +
  r4pde::theme_r4pde(font_size = 10)+
  scale_fill_whitebox_c(
    palette = "muted",
    labels = scales::label_number(suffix = "%"),
    n.breaks = 10,
    guide = guide_legend(reverse = TRUE)
  ) +
  labs(
    fill = "",
    title = "UAV multispectral mosaic",
    subtitle = "07.07.2021"
  )
```

Let's read all images collected at a later date:

```
# Open all images collected on 25th August 2021
tif1 <- "data/cercospora_data/multispec_2021_2022/1_r20210825.tif"
tif2 <- "data/cercospora_data/multispec_2021_2022/2_r20210825.tif"
tif3 <- "data/cercospora_data/multispec_2021_2022/3_r20210825.tif"
tif4 <- "data/cercospora_data/multispec_2021_2022/4_r20210825.tif"
tif5 <- "data/cercospora_data/multispec_2021_2022/5_r20210825.tif"
tif6 <- "data/cercospora_data/multispec_2021_2022/6_r20210825.tif"
tif7 <- "data/cercospora_data/multispec_2021_2022/7_r20210825.tif"
tif8 <- "data/cercospora_data/multispec_2021_2022/8_r20210825.tif"
tif9 <- "data/cercospora_data/multispec_2021_2022/9_r20210825.tif"
tif10 <- "data/cercospora_data/multispec_2021_2022/10_r20210825.tif"
```

```
p01 <- terra::rast(tif1) |> aggregate(20)
p02 <- terra::rast(tif2) |> aggregate(20)
p03 <- terra::rast(tif3) |> aggregate(20)
p04 <- terra::rast(tif4) |> aggregate(20)
p05 <- terra::rast(tif5) |> aggregate(20)
p06 <- terra::rast(tif6) |> aggregate(20)
p07 <- terra::rast(tif7) |> aggregate(20)
p08 <- terra::rast(tif8) |> aggregate(20)
p09 <- terra::rast(tif9) |> aggregate(20)
p10 <- terra::rast(tif10) |> aggregate(20)
```

```
# with many SpatRasters, make a SpatRasterCollection from a list
rlist <- list(p01, p02, p03, p04, p05, p06, p07, p08, p09, p10)
rsrc <- sprc(rlist)

mm <- merge(rsrc)
```

Warning: [merge] 9 raster(s) that did not share the base geometry of the first raster were resampled

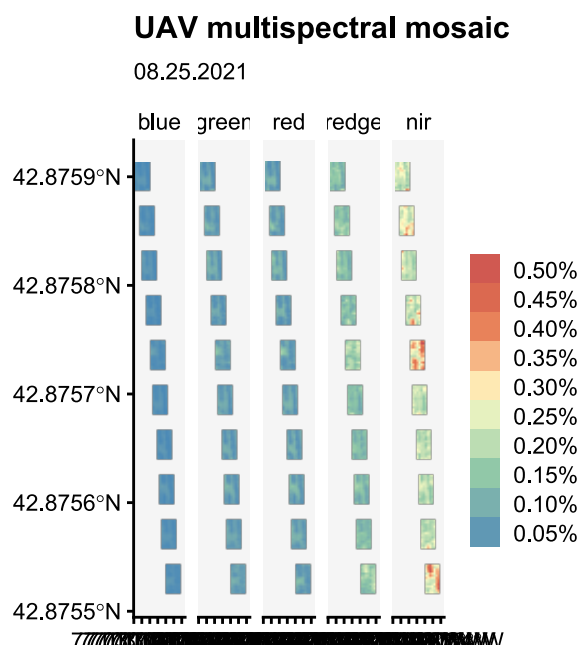
```
names(mm)
```

```
[1] "1_r20210825_1" "1_r20210825_2" "1_r20210825_3" "1_r20210825_4"
[5] "1_r20210825_5"
```

```
# Rename
mm2 <- mm |>
  rename(blue = "1_r20210825_1", green = "1_r20210825_2",
         red = "1_r20210825_3", redge = "1_r20210825_4",
         nir = "1_r20210825_5")
```

Now, a visualization task:

```
ggplot() +
  geom_spatraster(data = mm2) +
  facet_wrap(~lyr, ncol = 5) +
  r4pde::theme_r4pde(font_size = 10) +
  scale_fill_whitebox_c(
    palette = "muted",
    labels = scales::label_number(suffix = "%"),
    n.breaks = 10,
    guide = guide_legend(reverse = TRUE)
  ) +
  labs(
    fill = "",
    title = "UAV multispectral mosaic",
    subtitle = "08.25.2021"
  )
```



Reading severity data

Now, let's read the ground CLS data:

```
file <- "data/cercospora_data/CLS_DS.csv"
cls <- readr::read_csv(file, col_names = TRUE, show_col_types = FALSE)
```

What we got?

```
cls
```

```
# A tibble: 136 × 9
  Plot  D0  D1  D2  D3  D4  D5  D6  year
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1     1  0 0.25  0.9  4.58  8.75  21    NA  2021
2     2  0 0.75  3.68 12.4  20.8  45.4   NA  2021
3     3  0 2.92  7.38 18    34.6  65.2   NA  2021
4     4  0 0.75  5.45 17.4  42.6  75.5   NA  2021
5     5  0 3.38  3.45 15.4  17.2  55.2   NA  2021
6     6  0 0.8   1.38  8.8   12.2  15.4   NA  2021
7     7  0 0.375  3.05  9.1   13    31.0   NA  2021
8     8  0 3.7    8.4  29.4  41.2  71.8   NA  2021
9     9  0 2.38  4.28 13.2  15.6  38.1   NA  2021
10    10  0 2.22  2.28 30.2  17.3  68     NA  2021
# i 126 more rows
```

Note that columns D0 to D6 refer to the disease severity recorded in percentage at different dates for each study plot. Note also that this object is structured in *wide format* (i.e. each row represents a single plot for each year).

```
unique(cls$Plot)
```

```
[1] 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] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
[51] 51 52 53 54 55 56
```

Let's know the values stored in the *year* attribute:

```
unique(cls$year)
```

```
[1] 2021 2022 2023
```

Preparing a set of spectral covariables per plot & date

The 2021 data collection dates are 20210707, 20210715, 20210720, 20210802, and 20210825.

We will write a function that processes all the plot images of one date. For each image, the function reads the five bands, clamps the reflectance values to the range 0 to 1, computes several vegetation indices (VIs), and obtains the average value of the original bands and of the VIs. The results for all plots of the date are returned as a data frame with one row per plot.

```
process_date <- function(date_string) {
  # all plot images collected on the given date
  pattern <- paste0("*_r", format(as.Date(date_string), "%Y%m%d"), ".tif")
  files <- Sys.glob(file.path("data/cercospora_data/multispec_2021_2022", pattern))

  # plot number taken from the file name (files are sorted as text)
  plot_id <- as.integer(sub("_.*", "", basename(files)))
```

```

means <- lapply(files, function(file) {
  r <- clamp(rast(file), 0, 1)
  ndvi <- (r[[5]] - r[[3]]) / (r[[5]] + r[[3]])
  ndre <- (r[[5]] - r[[4]]) / (r[[5]] + r[[4]])
  cire <- (r[[5]] / r[[4]]) - 1
  sipi <- (r[[5]] - r[[1]]) / (r[[5]] - r[[3]])
  ndvi_range <- minmax(ndvi)
  ndvi_mean <- global(ndvi, fun = "mean", na.rm = TRUE)[, 1]
  mndvi <- 2 + log((ndvi - ndvi_mean) / (ndvi_range[2] - ndvi_range[1]))
  all_layers <- c(r, ndvi, ndre, cire, sipi, mndvi)
  global(all_layers, fun = "mean", na.rm = TRUE)[, 1]
})

df <- as.data.frame(do.call(rbind, means))
colnames(df) <- c("blue", "green", "red", "redge", "nir", "ndvi",
                 "ndre", "cire", "sipi", "mndvi")
df$plot <- plot_id
df$date <- as.Date(date_string)
df
}

```

Now we apply the function to the five dates of 2021 and write one CSV file per date (D0_2021.csv to D4_2021.csv). This step reads all the images, so it can take several minutes.

```

dates_2021 <- c(D0 = "2021-07-07", D1 = "2021-07-15", D2 = "2021-07-20",
               D3 = "2021-08-02", D4 = "2021-08-25")

for (i in seq_along(dates_2021)) {
  covariates <- process_date(dates_2021[[i]])
  write_csv(covariates,
            paste0("data/cercospora_data/", names(dates_2021)[i], "_2021.csv"))
}

```

Let's inspect the covariates of the first date:

```
read_csv("data/cercospora_data/D0_2021.csv", show_col_types = FALSE)
```

```

# A tibble: 40 × 12
   blue green red redge nir ndvi ndre cire sipi mndvi plot
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 0.0234 0.0596 0.0436 0.144 0.313 0.699 0.352 1.14 1.17 0.336 10
2 0.0228 0.0597 0.0443 0.143 0.300 0.691 0.340 1.07 1.18 0.383 11
3 0.0223 0.0603 0.0430 0.144 0.304 0.699 0.342 1.08 1.17 0.300 12
4 0.0221 0.0629 0.0427 0.149 0.318 0.712 0.348 1.11 1.16 0.364 13
5 0.0231 0.0599 0.0462 0.139 0.285 0.676 0.331 1.02 1.19 0.444 14
6 0.0224 0.0612 0.0415 0.146 0.312 0.720 0.350 1.12 1.15 0.249 15
7 0.0209 0.0587 0.0400 0.141 0.302 0.724 0.352 1.13 1.15 0.301 16
8 0.0213 0.0575 0.0399 0.137 0.296 0.716 0.355 1.14 1.15 0.341 17
9 0.0219 0.0587 0.0409 0.141 0.302 0.712 0.350 1.12 1.16 0.326 18
10 0.0226 0.0598 0.0436 0.143 0.299 0.696 0.340 1.07 1.18 0.398 19
# i 30 more rows
# i 1 more variable: date <date>

```

Put all covariate data in a single object

Now, we will read all the recently created csv files and put all covariate data values into a single object.

Next code obtains a list with the path of every csv file matching the D*_2021.csv expression:

```
(csv_files <- Sys.glob("data/cercospora_data/D*_2021.csv"))
```

```
[1] "data/cercospora_data/D0_2021.csv" "data/cercospora_data/D1_2021.csv"
[3] "data/cercospora_data/D2_2021.csv" "data/cercospora_data/D3_2021.csv"
[5] "data/cercospora_data/D4_2021.csv"
```

Now, we will read all files in the list:

```
(df.2021 <- do.call(rbind,lapply(csv_files,read.csv)))
```

	blue	green	red	redge	nir	ndvi	ndre
1	0.02343657	0.05959132	0.04362133	0.1444027	0.3131252	0.6988392	0.3515343
2	0.02282100	0.05970168	0.04427105	0.1432050	0.3003496	0.6909051	0.3404127
3	0.02229088	0.06034372	0.04296553	0.1441267	0.3037592	0.6985894	0.3424855
4	0.02211413	0.06289556	0.04265936	0.1493579	0.3175088	0.7124582	0.3477512
5	0.02311490	0.05985614	0.04618947	0.1392655	0.2849417	0.6763379	0.3314924
6	0.02237584	0.06117992	0.04150024	0.1462904	0.3117826	0.7198424	0.3504247
7	0.02086133	0.05870957	0.03996187	0.1412853	0.3017970	0.7235433	0.3521370
8	0.02132975	0.05750314	0.03989218	0.1370614	0.2956135	0.7157179	0.3550798
9	0.02188219	0.05869401	0.04093412	0.1413798	0.3023368	0.7115535	0.3503842
10	0.02260319	0.05975366	0.04357307	0.1429067	0.2992025	0.6962978	0.3402171
11	0.02089238	0.04880216	0.04060995	0.1215226	0.2446871	0.6569093	0.3221139
12	0.02350888	0.05890453	0.04759297	0.1368886	0.2807072	0.6557864	0.3276572
13	0.02218707	0.05638867	0.04349103	0.1341888	0.2819767	0.6780262	0.3441717
14	0.02275558	0.05684062	0.04544145	0.1339135	0.2753768	0.6666547	0.3337416
15	0.02272742	0.05738843	0.04389310	0.1379283	0.2873050	0.6838893	0.3418451
16	0.02140498	0.05666796	0.04005207	0.1370657	0.2937609	0.7116084	0.3544429
17	0.02086186	0.05736772	0.03807191	0.1398500	0.2991122	0.7290118	0.3560801
18	0.02015755	0.05423116	0.03848824	0.1311203	0.2787107	0.7109047	0.3537186
19	0.02193068	0.05637152	0.03996171	0.1306193	0.2834647	0.7008749	0.3578544
20	0.02214865	0.05725453	0.04177672	0.1331511	0.2830351	0.6914292	0.3488766
21	0.02122350	0.05473366	0.04193676	0.1295753	0.2737450	0.6830712	0.3463346
22	0.02200368	0.05443210	0.04510647	0.1327359	0.2787727	0.6671694	0.3387558
23	0.02334062	0.05691194	0.04427723	0.1309551	0.2608062	0.6588717	0.3212313
24	0.02211155	0.06021804	0.03976967	0.1403651	0.3081556	0.7156574	0.3615820
25	0.02174683	0.05562259	0.03816646	0.1312835	0.2858236	0.7057869	0.3560048
26	0.02082576	0.05881460	0.03393639	0.1407333	0.3136037	0.7548543	0.3704691
27	0.02192849	0.06039847	0.03532052	0.1438516	0.3201219	0.7514753	0.3694975
28	0.02302246	0.06453867	0.03590782	0.1454413	0.3360538	0.7627826	0.3862154
29	0.02060561	0.05634195	0.03435760	0.1348848	0.3139023	0.7577029	0.3915793
30	0.02453648	0.05776692	0.04101824	0.1400822	0.3165895	0.7123657	0.3729309
31	0.02449699	0.06516875	0.04320010	0.1488943	0.3369249	0.7188490	0.3734588
32	0.02375349	0.06403329	0.04276755	0.1462964	0.3370367	0.7210620	0.3804087
33	0.02255618	0.05711345	0.04411843	0.1383896	0.2971490	0.6880898	0.3481446
34	0.02217878	0.05772793	0.04388026	0.1337953	0.2885867	0.6798194	0.3490214
35	0.02187461	0.05617408	0.04287472	0.1365350	0.2917336	0.6953013	0.3481033
36	0.02296074	0.05817811	0.04563811	0.1391187	0.2921268	0.6801861	0.3393280
37	0.02267559	0.05957335	0.04213828	0.1420391	0.3082079	0.7142905	0.3568940
38	0.02360669	0.05999425	0.04294853	0.1433218	0.3103642	0.7052194	0.3528355
39	0.02364231	0.06191251	0.04298429	0.1483199	0.3220375	0.7163665	0.3560244

```

40 0.02427710 0.06139374 0.04394927 0.1444968 0.3085851 0.6985304 0.3476681
41 0.02052956 0.05020432 0.02973388 0.1338569 0.3185785 0.7882094 0.3981185
42 0.01859800 0.05052206 0.03105854 0.1318898 0.3058317 0.7805916 0.3922774
43 0.01981702 0.05348824 0.03288278 0.1322381 0.3022270 0.7638127 0.3819167
44 0.02000617 0.05614472 0.03134368 0.1402745 0.3260600 0.7887921 0.3904000
45 0.02095564 0.05658479 0.03321928 0.1439913 0.3291102 0.7828250 0.3835718
46 0.02237261 0.05968790 0.03353345 0.1488491 0.3470870 0.7903656 0.3917955
47 0.02100918 0.05998822 0.03180204 0.1560300 0.3643333 0.8096700 0.3941080
48 0.02181143 0.05995075 0.03243477 0.1516184 0.3587694 0.8000431 0.3988671
49 0.02132649 0.05706844 0.03089749 0.1471923 0.3448123 0.8004532 0.3926974
50 0.02159894 0.05753348 0.03114798 0.1416646 0.3302551 0.7919107 0.3914513
51 0.02038298 0.04942617 0.03205623 0.1217379 0.2899206 0.7603175 0.3967210
52 0.02266377 0.05830423 0.03712530 0.1481954 0.3369567 0.7581285 0.3783657
53 0.02015731 0.04875384 0.03535302 0.1201503 0.2717107 0.7356596 0.3799353
54 0.02165452 0.05001698 0.03766276 0.1215014 0.2670488 0.7155948 0.3650533
55 0.02265802 0.05245606 0.03529764 0.1328123 0.3006981 0.7540973 0.3802748
56 0.02051525 0.05213257 0.03268753 0.1382642 0.3256237 0.7833498 0.3957619
57 0.02128752 0.05645641 0.03279428 0.1483382 0.3501462 0.7989040 0.4000442
58 0.02095490 0.05585342 0.03271893 0.1444677 0.3498758 0.7997719 0.4113192
59 0.02185206 0.05682143 0.03272919 0.1417155 0.3463017 0.7920995 0.4109469
60 0.02202972 0.05714222 0.03436180 0.1461714 0.3441786 0.7825692 0.3957799
61 0.02098234 0.05483850 0.03241813 0.1348340 0.3203631 0.7814717 0.3997993
62 0.01901755 0.04792084 0.02924172 0.1168986 0.2677576 0.7610976 0.3815744
63 0.02388539 0.05884962 0.03938045 0.1462760 0.3194442 0.7426530 0.3626077
64 0.02140199 0.05705007 0.03633223 0.1389278 0.3217712 0.7609113 0.3886500
65 0.02201808 0.05953653 0.03894399 0.1374054 0.3171746 0.7461247 0.3899800
66 0.02075849 0.05788808 0.03145918 0.1467951 0.3700974 0.8162132 0.4289832
67 0.02015069 0.05689458 0.03134979 0.1515357 0.3765375 0.8211904 0.4239588
68 0.02026395 0.05835547 0.03137467 0.1573338 0.3931594 0.8284231 0.4268455
69 0.01992303 0.05484262 0.03180804 0.1509191 0.3793238 0.8212715 0.4326179
70 0.02065776 0.05741422 0.03358031 0.1470773 0.3617977 0.7967555 0.4185559
71 0.02220296 0.06168994 0.03496599 0.1541648 0.3697565 0.7967047 0.4086072
72 0.02038571 0.05698300 0.03187045 0.1479112 0.3529448 0.8037041 0.4055701
73 0.01967439 0.04903127 0.02905272 0.1239711 0.2874766 0.7748027 0.3851905
74 0.02137720 0.05585394 0.03576933 0.1306864 0.3214592 0.7630346 0.4159502
75 0.01963094 0.04868798 0.03043499 0.1312665 0.3107001 0.7820182 0.3978929
76 0.02113100 0.05204241 0.03232803 0.1335988 0.3020883 0.7639990 0.3756051
77 0.02048588 0.05105247 0.03119704 0.1411513 0.3349993 0.7918959 0.3975757
78 0.02243174 0.05674670 0.03130302 0.1461549 0.3674505 0.8055775 0.4201542
79 0.02187014 0.05383427 0.03043378 0.1401986 0.3415687 0.7977970 0.4083110
80 0.02188459 0.05444195 0.02963870 0.1365380 0.3385607 0.8036543 0.4163733
81 0.02108185 0.05315597 0.03629927 0.1615935 0.4061288 0.8110457 0.4222097
82 0.02107638 0.05629456 0.03690022 0.1572443 0.3762103 0.7876093 0.4011143
83 0.02178106 0.05907055 0.03867815 0.1628041 0.3881885 0.7835115 0.3986314
84 0.02047992 0.06098736 0.03665144 0.1691727 0.4158930 0.8096072 0.4135806
85 0.02129717 0.05862935 0.03736545 0.1607825 0.3928958 0.7985510 0.4127268
86 0.02194515 0.06127932 0.03643291 0.1634409 0.4002293 0.8077606 0.4146739
87 0.02097622 0.06260348 0.03316698 0.1759699 0.4279086 0.8375735 0.4146203
88 0.02082045 0.05968328 0.03347267 0.1692592 0.4194361 0.8302575 0.4193664
89 0.02052110 0.05855188 0.03285183 0.1667946 0.4183311 0.8334915 0.4252922
90 0.02118406 0.05952772 0.03409333 0.1669941 0.4164679 0.8290411 0.4224324
91 0.02066534 0.05198353 0.03630887 0.1469969 0.3602886 0.7819583 0.4100563
92 0.02286327 0.06025029 0.03771215 0.1674188 0.4105733 0.8133905 0.4146804
93 0.02418177 0.06143854 0.04705647 0.1632921 0.3503713 0.7270033 0.3567380
94 0.02357276 0.05738955 0.04634505 0.1583690 0.3343389 0.7189932 0.3482989
95 0.02251601 0.05564380 0.04242904 0.1626545 0.3609011 0.7562440 0.3706428
96 0.02082985 0.05493789 0.03747858 0.1581995 0.3747444 0.7851167 0.3976251
97 0.02073972 0.05667054 0.03645775 0.1673818 0.3937831 0.8025926 0.3977367

```

```

98 0.02024999 0.05755859 0.03581590 0.1662561 0.4091635 0.8133004 0.4166660
99 0.02082345 0.05814276 0.03632322 0.1663019 0.4077442 0.8067838 0.4127645
100 0.02130631 0.05952263 0.03733125 0.1691712 0.4058952 0.7983298 0.4039078
101 0.02079688 0.05889001 0.03666624 0.1697102 0.3959562 0.8020224 0.3937777
102 0.02154315 0.05398097 0.03832096 0.1526989 0.3727947 0.7756811 0.4090068
103 0.02068967 0.05666765 0.04165745 0.1675508 0.3665856 0.7608506 0.3641168
104 0.02287252 0.06036144 0.04135329 0.1691424 0.3596196 0.7572024 0.3523070
105 0.02327678 0.05737544 0.04043948 0.1639254 0.3273054 0.7462585 0.3248970
106 0.02055421 0.05506976 0.03165367 0.1679061 0.3830571 0.8245731 0.3848886
107 0.02045120 0.05556580 0.03147341 0.1721244 0.3911810 0.8288812 0.3848388
108 0.01936702 0.05359267 0.03041155 0.1670385 0.4003922 0.8384306 0.4065918
109 0.01892969 0.05400171 0.03181207 0.1714764 0.3980453 0.8328814 0.3948891
110 0.01908086 0.05353836 0.03287591 0.1710066 0.3968207 0.8205218 0.3920708
111 0.02038547 0.05592239 0.03392618 0.1781330 0.4067277 0.8204882 0.3854746
112 0.02008451 0.05720333 0.03208596 0.1750760 0.4149822 0.8331989 0.4011998
113 0.02067345 0.05449945 0.03739998 0.1559458 0.3971303 0.7953698 0.4269298
114 0.01985213 0.05373437 0.03486695 0.1637985 0.3662829 0.7934669 0.3733396
115 0.02132429 0.05593877 0.03814748 0.1559295 0.3894839 0.7920895 0.4204294
116 0.02137609 0.05544646 0.03941156 0.1535504 0.3745666 0.7752996 0.4094564
117 0.02173253 0.05606452 0.03581073 0.1600026 0.3982399 0.8091809 0.4198104
118 0.01983960 0.05239147 0.03299102 0.1556517 0.3980271 0.8196374 0.4295256
119 0.01958886 0.05078259 0.03284344 0.1529905 0.3931337 0.8197849 0.4326471
120 0.02143953 0.05433169 0.03362620 0.1559428 0.4089219 0.8242763 0.4413151
121 0.02456960 0.05829370 0.03337980 0.2035489 0.4674467 0.8522272 0.3940106
122 0.02227492 0.05626306 0.03402620 0.1924762 0.4087379 0.8252356 0.3618694
123 0.02523303 0.06403575 0.03893872 0.1969021 0.4110210 0.8061321 0.3530707
124 0.02431507 0.06981933 0.03149349 0.2080660 0.4400168 0.8538962 0.3608609
125 0.02721698 0.06959043 0.04073738 0.2240817 0.4563009 0.8229697 0.3463280
126 0.02798405 0.07293051 0.04063069 0.2243524 0.4512955 0.8183378 0.3413659
127 0.02362352 0.06819463 0.03388499 0.2225483 0.4584950 0.8516509 0.3536612
128 0.02337971 0.06235514 0.03212167 0.2017982 0.4440984 0.8520046 0.3771044
129 0.02266283 0.06217227 0.03157511 0.2027005 0.4475631 0.8549204 0.3785745
130 0.02366451 0.06000658 0.03321898 0.1894632 0.4401444 0.8441077 0.4001559
131 0.02513064 0.06334645 0.03448836 0.1972631 0.4274775 0.8350754 0.3716506
132 0.02377771 0.05588092 0.03341388 0.1794289 0.4061702 0.8257592 0.3845869
133 0.02449490 0.05809191 0.04298490 0.1784245 0.3147830 0.7299304 0.2795985
134 0.02731799 0.05988043 0.04708922 0.1774982 0.3350545 0.7195767 0.3055429
135 0.02619083 0.05744756 0.03728376 0.1842603 0.3627074 0.7868434 0.3266340
136 0.02581965 0.06053502 0.03775902 0.1914101 0.3971750 0.8019641 0.3518380
137 0.02561499 0.06698112 0.04009003 0.2271990 0.4421288 0.8138778 0.3278621
138 0.02400703 0.06453540 0.03525491 0.2070071 0.4443489 0.8351464 0.3707701
139 0.02158047 0.05805170 0.02776623 0.1873562 0.4307834 0.8663875 0.3984368
140 0.02182744 0.05941715 0.02958924 0.1884719 0.4182472 0.8503351 0.3807011
141 0.02015692 0.05312046 0.02761516 0.1659301 0.3789922 0.8446382 0.3930457
142 0.02510567 0.06159803 0.03564669 0.1842460 0.3929318 0.8109585 0.3607708
143 0.01993199 0.04180365 0.02775356 0.1207182 0.2949298 0.8009226 0.4183965
144 0.02300851 0.05705218 0.03661990 0.1733665 0.3311235 0.7724715 0.3138201
145 0.02476484 0.05523785 0.03972681 0.1718589 0.3296071 0.7534944 0.3139303
146 0.02154290 0.05242739 0.03053955 0.1765510 0.3719669 0.8277281 0.3569453
147 0.02210420 0.05258662 0.03051192 0.1727336 0.3759095 0.8276559 0.3698586
148 0.02330147 0.05844441 0.03332284 0.1898374 0.4210377 0.8347262 0.3800209
149 0.02282903 0.05852711 0.03503826 0.2062768 0.4417818 0.8371609 0.3690559
150 0.02201556 0.05756078 0.03422850 0.1881062 0.4182142 0.8332135 0.3817618
151 0.02389892 0.05898878 0.03347903 0.1847862 0.4078247 0.8310272 0.3799784
152 0.02083712 0.05540730 0.02953161 0.1768605 0.3941414 0.8413166 0.3825045
153 0.02458310 0.06176598 0.03476932 0.1976596 0.4174477 0.8280028 0.3586799
154 0.01718078 0.04930132 0.02856733 0.1486271 0.3652903 0.8334064 0.4221488
155 0.02748465 0.06946837 0.04363258 0.2318181 0.4706404 0.8123663 0.3418064

```

```

156 0.02747831 0.07332335 0.04389518 0.2264001 0.4458781 0.7968273 0.3272567
157 0.02521912 0.06154772 0.03600436 0.2019781 0.4287934 0.8282891 0.3619903
158 0.02434867 0.06283113 0.03451642 0.2081286 0.4614013 0.8472470 0.3825518
159 0.02530022 0.05947328 0.03604093 0.2065795 0.4364578 0.8318662 0.3616867
160 0.02323038 0.06380880 0.03683221 0.2106691 0.4795214 0.8422059 0.3925382
161 0.03244730 0.06243503 0.07069395 0.1331881 0.2240197 0.5263451 0.2603764
162 0.03220764 0.06421227 0.07358433 0.1305611 0.2126103 0.4872965 0.2416332
163 0.03148392 0.06400177 0.06896637 0.1465971 0.2389044 0.5390938 0.2399182
164 0.03167203 0.06808206 0.06547172 0.1751177 0.3086365 0.6280323 0.2732817
165 0.03320107 0.06398255 0.07626774 0.1272121 0.1949513 0.4500080 0.2154965
166 0.03155779 0.06765107 0.06388105 0.1640356 0.2689753 0.6018466 0.2430515
167 0.03049306 0.06404963 0.06585743 0.1490913 0.2448651 0.5740527 0.2454515
168 0.03252230 0.06384611 0.06648974 0.1375281 0.2243945 0.5383073 0.2420900
169 0.02987872 0.05902407 0.06794822 0.1251337 0.2040113 0.5055839 0.2431990
170 0.03446312 0.06983649 0.07757790 0.1447033 0.2299757 0.5042939 0.2310344
171 0.03317173 0.06795603 0.06613100 0.1632761 0.3060204 0.6192177 0.2999994
172 0.03227812 0.06842014 0.07143898 0.1660777 0.2665647 0.5637743 0.2309203
173 0.03398287 0.07115492 0.07250453 0.1389441 0.2177167 0.5067603 0.2260623
174 0.02865669 0.06429306 0.05747477 0.1574716 0.2590684 0.6237297 0.2431701
175 0.03021290 0.06066011 0.06666227 0.1384870 0.2170173 0.5271643 0.2225675
176 0.02967587 0.06158339 0.06095244 0.1500869 0.2465646 0.5848673 0.2416536
177 0.02884645 0.05705891 0.06533360 0.1202913 0.1869982 0.4882258 0.2250775
178 0.03276704 0.06359781 0.07456907 0.1251366 0.1891989 0.4570189 0.2115906
179 0.02708242 0.05999907 0.05766794 0.1504990 0.2450337 0.5946147 0.2374369
180 0.02766003 0.05688758 0.06456498 0.1280580 0.2001966 0.5069556 0.2206031
181 0.02838966 0.05722613 0.06488701 0.1271373 0.1992304 0.5115626 0.2244217
182 0.03559652 0.06614677 0.07592767 0.1315287 0.2143333 0.4807597 0.2449726
183 0.03122141 0.06389911 0.07011235 0.1232556 0.1820245 0.4541211 0.1994603
184 0.02919906 0.06398205 0.06071808 0.1454490 0.2416222 0.5913800 0.2485037
185 0.02982668 0.06742336 0.06261795 0.1664833 0.2768481 0.6132203 0.2448847
186 0.02978697 0.05961399 0.06366181 0.1285860 0.2094327 0.5358823 0.2420457
187 0.02802117 0.05635194 0.05964353 0.1242803 0.2039935 0.5432331 0.2473246
188 0.02804431 0.05990398 0.05714082 0.1520841 0.2605731 0.6137755 0.2602729
189 0.03115430 0.06222985 0.06680599 0.1226242 0.1971786 0.5040743 0.2421152
190 0.03017063 0.05987192 0.06688301 0.1147182 0.1808972 0.4625647 0.2312153
191 0.02969164 0.06141537 0.06461569 0.1333502 0.2173871 0.5265676 0.2410969
192 0.03245863 0.06580556 0.07369124 0.1179781 0.1793431 0.4283260 0.2142952
193 0.03780640 0.06896099 0.08072512 0.1249145 0.2023160 0.4538598 0.2488864
194 0.03015803 0.06140500 0.06845817 0.1301699 0.2018116 0.4929597 0.2222330
195 0.03273967 0.06385214 0.07184862 0.1220430 0.2069285 0.4991207 0.2646952
196 0.03340377 0.06654732 0.07474162 0.1302466 0.2149186 0.4934244 0.2531492
197 0.03576925 0.07247605 0.07037803 0.1731286 0.3140637 0.6212680 0.2859256
198 0.02975048 0.05792985 0.06628184 0.1342174 0.2345209 0.5560936 0.2760421
199 0.03765677 0.07335121 0.08514252 0.1333163 0.2088120 0.4441819 0.2313678
200 0.03487981 0.06780985 0.07374642 0.1484053 0.2625147 0.5622498 0.2799192

    cire      sipi      mndvi plot      date
1  1.1415594 1.168504 0.3357828097 10 2021-07-07
2  1.0743886 1.178469 0.3831091052 11 2021-07-07
3  1.0801215 1.171007 0.3004619363 12 2021-07-07
4  1.1075915 1.160622 0.3637140071 13 2021-07-07
5  1.0232721 1.188881 0.4442368504 14 2021-07-07
6  1.1150448 1.146710 0.2486349703 15 2021-07-07
7  1.1250963 1.147409 0.3010096154 16 2021-07-07
8  1.1420617 1.153974 0.3411934009 17 2021-07-07
9  1.1186718 1.158002 0.3256349981 18 2021-07-07
10 1.0709430 1.177161 0.3981083214 19 2021-07-07
11 0.9989965 1.208993 0.4992914029 1 2021-07-07
12 1.0246774 1.222624 0.5812196925 20 2021-07-07

```


13	1.0917440	1.187306	0.5261465375	21	2021-07-07
14	1.0412629	1.199819	0.5088588230	22	2021-07-07
15	1.0772731	1.179034	0.3970453680	23	2021-07-07
16	1.1437437	1.152637	0.4036515029	24	2021-07-07
17	1.1446112	1.135257	0.1922804965	25	2021-07-07
18	1.1474183	1.154173	0.3741285225	26	2021-07-07
19	1.1670434	1.164372	0.2192195769	27	2021-07-07
20	1.1211586	1.180103	0.2670115421	28	2021-07-07
21	1.1068452	1.183483	0.3921222267	29	2021-07-07
22	1.0722585	1.202996	0.4383503430	2	2021-07-07
23	1.0088580	1.212415	0.3244785295	30	2021-07-07
24	1.1826306	1.145451	0.3773438604	31	2021-07-07
25	1.1565180	1.151022	0.3542278410	32	2021-07-07
26	1.2309874	1.106173	0.1469966333	33	2021-07-07
27	1.2197816	1.108914	0.0840190302	34	2021-07-07
28	1.3102119	1.094956	0.0132994128	35	2021-07-07
29	1.3468557	1.104615	0.0978333987	36	2021-07-07
30	1.2481819	1.141349	0.3183810870	37	2021-07-07
31	1.2416056	1.144968	0.3515716385	38	2021-07-07
32	1.2898444	1.144735	0.3309417213	39	2021-07-07
33	1.1165162	1.181832	0.4072417552	3	2021-07-07
34	1.1218080	1.190335	0.4463134659	40	2021-07-07
35	1.1116724	1.169715	0.3786825647	4	2021-07-07
36	1.0754732	1.183114	0.4450529648	5	2021-07-07
37	1.1581654	1.149740	0.3065144945	6	2021-07-07
38	1.1404401	1.158901	0.2235725778	7	2021-07-07
39	1.1538314	1.148876	0.2117349834	8	2021-07-07
40	1.1171347	1.163275	0.2623829017	9	2021-07-07
41	1.3782598	1.071387	0.0205864548	10	2021-07-15
42	1.3566259	1.090718	0.0250170201	11	2021-07-15
43	1.2891365	1.100060	0.0061431869	12	2021-07-15
44	1.3329724	1.079262	-0.0686704756	13	2021-07-15
45	1.2906723	1.080714	-0.0450803877	14	2021-07-15
46	1.3350544	1.073452	-0.1398443835	15	2021-07-15
47	1.3428199	1.063935	-0.3200177751	16	2021-07-15
48	1.3698651	1.068173	-0.1140547795	17	2021-07-15
49	1.3478594	1.066137	-0.1478313020	18	2021-07-15
50	1.3304157	1.069318	-0.1254230424	19	2021-07-15
51	1.3809846	1.095432	0.1445930197	1	2021-07-15
52	1.2723543	1.104388	0.1423226946	20	2021-07-15
53	1.2872960	1.120003	0.2427405562	21	2021-07-15
54	1.1995585	1.131178	0.2966248439	22	2021-07-15
55	1.2711511	1.095153	-0.0224570025	23	2021-07-15
56	1.3672315	1.081331	0.0539492552	24	2021-07-15
57	1.3914122	1.068620	-0.1999835409	25	2021-07-15
58	1.4482764	1.069124	-0.0812479746	26	2021-07-15
59	1.4574950	1.071044	-0.2049412495	27	2021-07-15
60	1.3665155	1.081635	-0.1793357879	28	2021-07-15
61	1.3814683	1.078394	-0.1324388877	29	2021-07-15
62	1.2858749	1.089837	0.0458092649	2	2021-07-15
63	1.2022264	1.116328	0.0183683589	30	2021-07-15
64	1.3347470	1.102757	0.1305738483	31	2021-07-15
65	1.3490326	1.109362	0.0378045532	32	2021-07-15
66	1.5758712	1.057704	-0.3042757279	33	2021-07-15
67	1.5293288	1.059005	-0.3881285964	34	2021-07-15
68	1.5529622	1.055250	-0.3992933887	35	2021-07-15
69	1.6024960	1.060230	-0.3792228663	36	2021-07-15
70	1.5128737	1.077600	-0.0678144716	37	2021-07-15

71	1.4460478	1.075913	-0.1745530817	38	2021-07-15
72	1.4243774	1.070826	-0.2386634167	39	2021-07-15
73	1.3121340	1.080626	0.0282156325	3	2021-07-15
74	1.5050103	1.102703	0.0438634149	40	2021-07-15
75	1.3785072	1.079370	0.0274106440	4	2021-07-15
76	1.2548895	1.087083	0.1502160095	5	2021-07-15
77	1.3748307	1.074167	-0.0794766912	6	2021-07-15
78	1.5042189	1.058518	-0.2078983833	7	2021-07-15
79	1.4313024	1.060803	-0.1346557992	8	2021-07-15
80	1.4784583	1.055124	-0.2510148625	9	2021-07-15
81	1.5091684	1.065504	-0.1696912521	10	2021-07-20
82	1.3854419	1.085917	-0.0483153431	11	2021-07-20
83	1.3605557	1.087327	-0.0953645846	12	2021-07-20
84	1.4496963	1.074684	-0.1508310510	13	2021-07-20
85	1.4437110	1.076338	-0.1012477650	14	2021-07-20
86	1.4500552	1.068223	-0.2495194123	15	2021-07-20
87	1.4430745	1.050196	-0.5378942151	16	2021-07-20
88	1.4756373	1.054488	-0.3193034281	17	2021-07-20
89	1.5209923	1.051677	-0.3029425461	18	2021-07-20
90	1.4934348	1.053301	-0.3581289293	19	2021-07-20
91	1.4544218	1.090791	0.0765274948	1	2021-07-20
92	1.4523725	1.056664	-0.1852615254	20	2021-07-20
93	1.1368177	1.130660	0.2412537373	21	2021-07-20
94	1.0923325	1.135407	0.2240251204	22	2021-07-20
95	1.2074760	1.108039	-0.0651335365	23	2021-07-20
96	1.3581943	1.088449	0.0090607709	24	2021-07-20
97	1.3492680	1.074566	-0.2128007022	25	2021-07-20
98	1.4664407	1.070300	-0.2144020219	26	2021-07-20
99	1.4447817	1.075076	-0.2931720787	27	2021-07-20
100	1.3892259	1.080070	-0.0064584782	28	2021-07-20
101	1.3287352	1.076244	-0.1350575729	29	2021-07-20
102	1.4399733	1.093076	0.1324019681	2	2021-07-20
103	1.1796431	1.110340	-0.0759637430	30	2021-07-20
104	1.1143660	1.106180	-0.1119407775	31	2021-07-20
105	0.9838179	1.098887	-0.0819634859	32	2021-07-20
106	1.2786321	1.050752	-0.3987374946	33	2021-07-20
107	1.2778733	1.050203	-0.4709627746	34	2021-07-20
108	1.3963778	1.047230	-0.5005099302	35	2021-07-20
109	1.3281949	1.052344	-0.4394932170	36	2021-07-20
110	1.3181372	1.062573	-0.1964393774	37	2021-07-20
111	1.2844313	1.061355	-0.3757807604	38	2021-07-20
112	1.3688477	1.052621	-0.4935354112	39	2021-07-20
113	1.5577239	1.084939	0.0619995742	3	2021-07-20
114	1.2300633	1.080936	-0.0512569619	40	2021-07-20
115	1.4995324	1.082575	0.0467589503	4	2021-07-20
116	1.4469041	1.095501	0.1295138163	5	2021-07-20
117	1.4935743	1.069802	-0.1673312381	6	2021-07-20
118	1.5547877	1.065488	-0.2913712994	7	2021-07-20
119	1.5787545	1.062266	-0.2423790191	8	2021-07-20
120	1.6324658	1.056176	-0.4109303033	9	2021-07-20
121	1.3324811	1.031138	-0.9329644398	10	2021-08-02
122	1.1790729	1.053371	-0.4702134438	11	2021-08-02
123	1.1274064	1.062791	-0.4919324211	12	2021-08-02
124	1.1642643	1.030428	-0.9219435727	13	2021-08-02
125	1.0878222	1.058747	-0.8734865582	14	2021-08-02
126	1.0703626	1.050969	-0.4580022808	15	2021-08-02
127	1.1245460	1.036121	-0.8471248176	16	2021-08-02
128	1.2452050	1.035747	-0.8700951548	17	2021-08-02

129	1.2534928	1.035822	-0.7559168262	18	2021-08-02
130	1.3741697	1.042222	-1.0814628753	19	2021-08-02
131	1.2310151	1.041681	-0.5201193156	1	2021-08-02
132	1.2930730	1.047657	-0.4141809408	20	2021-08-02
133	0.8095930	1.115223	0.0056911166	21	2021-08-02
134	0.9136160	1.124892	-0.0245878304	22	2021-08-02
135	1.0016757	1.064336	-0.4527557321	23	2021-08-02
136	1.1278397	1.061822	-0.7375545901	24	2021-08-02
137	1.0184010	1.056483	-0.5740803039	25	2021-08-02
138	1.2284968	1.044680	-0.7155234903	26	2021-08-02
139	1.3590637	1.024674	-1.0858622822	27	2021-08-02
140	1.2615503	1.034384	-0.7847178421	28	2021-08-02
141	1.3249284	1.035002	-0.7231518295	29	2021-08-02
142	1.1773176	1.054953	-0.4577592358	2	2021-08-02
143	1.5202302	1.051947	-0.4565179757	30	2021-08-02
144	0.9398590	1.076097	-0.2878019011	31	2021-08-02
145	0.9409901	1.084722	-0.2876957587	32	2021-08-02
146	1.1373376	1.042886	-0.6047624726	33	2021-08-02
147	1.2034997	1.049083	-0.9068635437	34	2021-08-02
148	1.2560443	1.043206	-0.8599549281	35	2021-08-02
149	1.2022205	1.045737	-0.9818120336	36	2021-08-02
150	1.2628642	1.056609	-0.9147740655	37	2021-08-02
151	1.2545735	1.044365	-0.8599796085	38	2021-08-02
152	1.2602425	1.039036	-0.5206771201	39	2021-08-02
153	1.1628946	1.045994	-0.5744413976	3	2021-08-02
154	1.5433084	1.053258	-0.6361958420	40	2021-08-02
155	1.0764829	1.063901	-0.5624987735	4	2021-08-02
156	1.0193597	1.074447	-0.3103782430	5	2021-08-02
157	1.1817977	1.046306	-0.6498980336	6	2021-08-02
158	1.2882047	1.039040	-0.8562301124	7	2021-08-02
159	1.1714568	1.043690	-0.7183310744	8	2021-08-02
160	1.3420360	1.049136	-0.7612609133	9	2021-08-02
161	0.7298258	1.324342	0.0698481711	10	2021-08-25
162	0.6602580	1.393986	0.1635182319	11	2021-08-25
163	0.6482139	1.304943	0.0540153634	12	2021-08-25
164	0.7777530	1.215610	0.0089946689	13	2021-08-25
165	0.5605548	1.425463	-0.2514779990	14	2021-08-25
166	0.6562813	1.223703	-0.0269560659	15	2021-08-25
167	0.6699439	1.264781	0.0280182172	16	2021-08-25
168	0.6564857	1.298317	0.0010159223	17	2021-08-25
169	0.6654889	1.360910	0.0434079012	18	2021-08-25
170	0.6200022	1.385264	0.2062654531	19	2021-08-25
171	0.9131850	1.225855	0.0860920107	1	2021-08-25
172	0.6216048	1.304122	0.1443419035	20	2021-08-25
173	0.5981404	1.337526	-0.1961771307	21	2021-08-25
174	0.6564541	1.196577	-0.1139408403	22	2021-08-25
175	0.5855172	1.316696	-0.1561919432	23	2021-08-25
176	0.6572801	1.270045	0.1313856963	24	2021-08-25
177	0.6019302	1.370359	-0.0051956405	25	2021-08-25
178	0.5535844	1.444511	-0.0308850317	26	2021-08-25
179	0.6428781	1.260062	0.0416937059	27	2021-08-25
180	0.5872713	1.376995	0.1208025802	28	2021-08-25
181	0.5958389	1.351988	-0.0581929352	29	2021-08-25
182	0.6723947	1.366653	0.0264775382	2	2021-08-25
183	0.5166089	1.415353	-0.0897205516	30	2021-08-25
184	0.6779860	1.231409	0.0007949469	31	2021-08-25
185	0.6634278	1.216019	-0.0764145067	32	2021-08-25
186	0.6564104	1.294261	-0.0502768648	33	2021-08-25

```

187 0.6858638 1.292920 0.0990268744 34 2021-08-25
188 0.7262786 1.226937 0.0094243580 35 2021-08-25
189 0.6753764 1.339246 0.0401994088 36 2021-08-25
190 0.6312105 1.419802 0.1535082537 37 2021-08-25
191 0.6580441 1.335205 0.0355454843 38 2021-08-25
192 0.5638364 1.474723 0.0133503191 39 2021-08-25
193 0.6897908 1.399921 0.0388698356 3 2021-08-25
194 0.6005139 1.374915 -0.0782489060 40 2021-08-25
195 0.7484466 1.346672 0.0512449521 4 2021-08-25
196 0.7062528 1.353491 0.0458086211 5 2021-08-25
197 0.8448268 1.237137 0.1118462609 6 2021-08-25
198 0.7923682 1.295075 0.0697294939 7 2021-08-25
199 0.6194196 1.437227 0.1443525005 8 2021-08-25
200 0.8111978 1.278937 0.1160957419 9 2021-08-25

```

Then, we will convert the dataframe into a tibble:

```
covar.2021 <- tibble(df.2021)
```

Let's check what we got:

```
covar.2021
```

```

# A tibble: 200 × 12
   blue green red redge nir ndvi ndre cire sipi mndvi plot date
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int> <chr>
1 0.0234 0.0596 0.0436 0.144 0.313 0.699 0.352 1.14 1.17 0.336 10 2021-07...
2 0.0228 0.0597 0.0443 0.143 0.300 0.691 0.340 1.07 1.18 0.383 11 2021-07...
3 0.0223 0.0603 0.0430 0.144 0.304 0.699 0.342 1.08 1.17 0.300 12 2021-07...
4 0.0221 0.0629 0.0427 0.149 0.318 0.712 0.348 1.11 1.16 0.364 13 2021-07...
5 0.0231 0.0599 0.0462 0.139 0.285 0.676 0.331 1.02 1.19 0.444 14 2021-07...
6 0.0224 0.0612 0.0415 0.146 0.312 0.720 0.350 1.12 1.15 0.249 15 2021-07...
7 0.0209 0.0587 0.0400 0.141 0.302 0.724 0.352 1.13 1.15 0.301 16 2021-07...
8 0.0213 0.0575 0.0399 0.137 0.296 0.716 0.355 1.14 1.15 0.341 17 2021-07...
9 0.0219 0.0587 0.0409 0.141 0.302 0.712 0.350 1.12 1.16 0.326 18 2021-07...
10 0.0226 0.0598 0.0436 0.143 0.299 0.696 0.340 1.07 1.18 0.398 19 2021-07...
# i 190 more rows

```

Now, we will need to join the CLS severity data to the spectral covariates data.

Let's recall how the CLS data are structured:

```
cls
```

```

# A tibble: 136 × 9
   Plot D0 D1 D2 D3 D4 D5 D6 year
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 1 0 0.25 0.9 4.58 8.75 21 NA 2021
2 2 0 0.75 3.68 12.4 20.8 45.4 NA 2021
3 3 0 2.92 7.38 18 34.6 65.2 NA 2021
4 4 0 0.75 5.45 17.4 42.6 75.5 NA 2021
5 5 0 3.38 3.45 15.4 17.2 55.2 NA 2021
6 6 0 0.8 1.38 8.8 12.2 15.4 NA 2021
7 7 0 0.375 3.05 9.1 13 31.0 NA 2021
8 8 0 3.7 8.4 29.4 41.2 71.8 NA 2021
9 9 0 2.38 4.28 13.2 15.6 38.1 NA 2021

```

```
10    10    0 2.22  2.28 30.2 17.3  68    NA  2021
# i 126 more rows
```

Note that the `covar.2021` object has a date attribute in a single column and that the `cls` object has different columns represent dates (i.e. D0 to D5). This means that we need to change the `cls` object structure to be able to join its attributes to the `covar.2021` object. In technical terms, our task is to convert the `cls` table from *wide* format into *long* format.

We will use the `pivot_longer` function from the `tidyr` package:

```
cls |>
  pivot_longer(
    cols = starts_with("D"),
    names_to = "date",
    values_to = "severity",
    values_drop_na = TRUE
  ) -> long_cls
```

Now, we will do an additional step, we will replace qualitative date values (D0, D1, ...) for actual date values.

```
change <- tibble(old = c("D0", "D1", "D2", "D3", "D4"),
                 new = c("2021-07-07", "2021-07-15", "2021-07-20",
                        "2021-08-02", "2021-08-25"))
```

Let's check what we got:

```
change
```

```
# A tibble: 5 × 2
  old   new
<chr> <chr>
1 D0    2021-07-07
2 D1    2021-07-15
3 D2    2021-07-20
4 D3    2021-08-02
5 D4    2021-08-25
```

Now, let's do a left join to produce the changes:

```
(ncls <- left_join(long_cls, change, by = c(date = "old")))
```

```
# A tibble: 872 × 5
  Plot year date severity new
<dbl> <dbl> <chr>    <dbl> <chr>
1     1  2021 D0         0  2021-07-07
2     1  2021 D1        0.25 2021-07-15
3     1  2021 D2         0.9 2021-07-20
4     1  2021 D3        4.58 2021-08-02
5     1  2021 D4        8.75 2021-08-25
6     1  2021 D5        21    <NA>
7     2  2021 D0         0  2021-07-07
8     2  2021 D1        0.75 2021-07-15
9     2  2021 D2        3.68 2021-07-20
```

```
10      2  2021 D3      12.4  2021-08-02
# i 862 more rows
```

Finally, we can apply a join:

```
ncovar.2021 <- left_join(ncIs, covar.2021, by = c("Plot" = "plot", "new" = "date"))
```

```
ncovar.2021
```

```
# A tibble: 872 × 15
  Plot year date severity new      blue green      red redge  nir  ndvi
<dbl> <dbl> <chr>    <dbl> <chr>    <dbl> <dbl>    <dbl> <dbl>  <dbl> <dbl>
1     1     1  2021 D0         0  2021...  0.0209  0.0488  0.0406  0.122  0.245  0.657
2     2     1  2021 D1        0.25  2021...  0.0204  0.0494  0.0321  0.122  0.290  0.760
3     3     1  2021 D2        0.9  2021...  0.0207  0.0520  0.0363  0.147  0.360  0.782
4     4     1  2021 D3        4.58  2021...  0.0251  0.0633  0.0345  0.197  0.427  0.835
5     5     1  2021 D4        8.75  2021...  0.0332  0.0680  0.0661  0.163  0.306  0.619
6     6     1  2021 D5        21  <NA>    NA      NA      NA      NA      NA      NA
7     7     2  2021 D0         0  2021...  0.0220  0.0544  0.0451  0.133  0.279  0.667
8     8     2  2021 D1        0.75  2021...  0.0190  0.0479  0.0292  0.117  0.268  0.761
9     9     2  2021 D2        3.68  2021...  0.0215  0.0540  0.0383  0.153  0.373  0.776
10    10     2  2021 D3       12.4  2021...  0.0251  0.0616  0.0356  0.184  0.393  0.811
# i 862 more rows
# i 4 more variables: ndre <dbl>, cire <dbl>, sipi <dbl>, mndvi <dbl>
```

Change severity percentage to severity categories

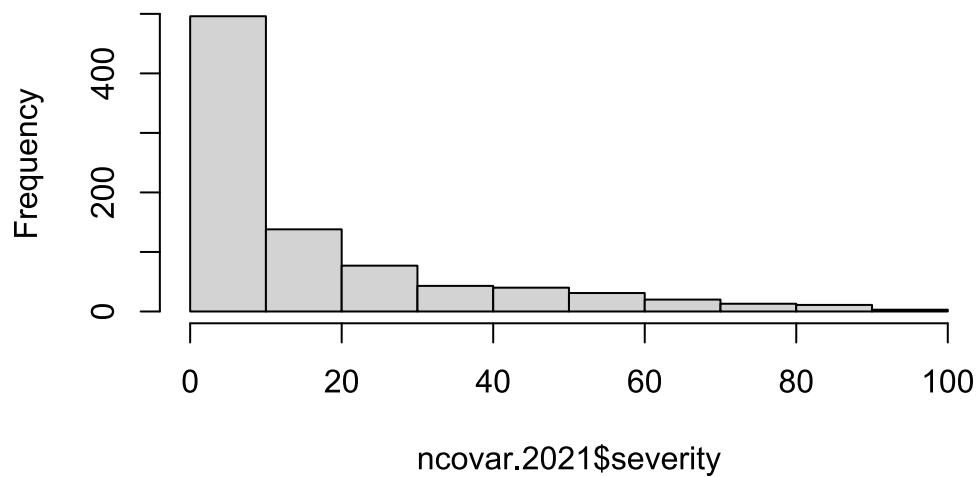
Our final step is to convert percentage severity values into ordinal severity levels.

First, let's check the severity histogram

```
#uncomment if you need to the read the file
#ncovar.2021 <- read_csv("./data/ncovar_2021.csv", show_col_types = FALSE)
```

```
hist(ncovar.2021$severity)
```

Histogram of ncovar.2021\$severity



Next code produces seven levels of CLS severity:

Severity [%]	Category
0 - 1	0
1 - 10	1
10 - 20	2
20 - 30	3
30 - 40	4
40 - 50	5
50 - 100	6

```
fcovar.2021 <- ncovar.2021 |> mutate(category=cut(severity, breaks=c(0,1, 10, 20, 30,40, 50, 100),
labels=c(0,1,2,3,4,5,6), include.lowest = TRUE))
```

What we got?

```
summary(fcovar.2021$category)
```

```
 0   1   2   3   4   5   6
284 212 138  77  43  40  78
```

Note that the distribution of values at each disease level is not balanced.

Write 2021 csv files

```
write_csv(
  ncovar.2021,
  "data/cercospora_data/ncovar_2021.csv",
  na = "NA",
  col_names = TRUE,
```

```
  append = FALSE
)
```

```
write_csv(
  fcovar.2021,
  "data/cercospora_data/fcovar_2021.csv",
  na = "NA",
  col_names = TRUE,
  append = FALSE
)
```

In this first part we have conducted a very important task: to translate UAV multispectral imagery and CLS severity ground data into a single table storing spectral covariates and severity classes which we will use next for building a prediction model.

Machine learning model

Now we will train & test a machine learning model to estimate qualitative levels of CLS disease severity using the previously created dataset.

Software setup

Let's start by cleaning up R memory:

```
rm(list=ls())
```

Then, we need to install several packages (if they are not installed yet):

```
list.of.packages <- c("readr", "terra", "tidyterra", "stars", "sf", "leaflet", "leafem", "dplyr", "ggplot2",
"tidymodels", "xgboost", "tune")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)
```

Now, let's load all the required packages:

```
library(readr)
library(terra)
library(tidyterra)
library(stars)
library(sf)
library(leaflet)
library(leafem)
library(dplyr)
library(ggplot2)
library(tidymodels)
library(xgboost)
library(tune)
```

What files are in the data folder under the current path:

```
list.files("data/cercospora_data")
```

```
[1] "CLS_DS.csv"      "D0_2021.csv"    "D1_2021.csv"
[4] "D2_2021.csv"    "D3_2021.csv"    "D4_2021.csv"
[7] "fcovar_2021.csv" "multispec_2021_2022" "multispec_2023"
[10] "ncovar_2021.csv" "README.md"
```


Let's remind that *fcovar_2021.csv* is the file containing the dataset we created previously. We will use the *readr* package to read the data as a tibble:

```
covar21 <- read_csv("data/cercospora_data/fcovar_2021.csv",
                    col_types = cols(category = col_character())) |>
  drop_na()
```

What we got?

```
covar21
```

```
# A tibble: 600 × 16
  Plot year date severity new blue green red redge nir ndvi
<dbl> <dbl> <chr> <dbl> <date> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 1 2021 D0 0 2021-07-07 0.0209 0.0488 0.0406 0.122 0.245 0.657
2 1 2021 D1 0.25 2021-07-15 0.0204 0.0494 0.0321 0.122 0.290 0.760
3 1 2021 D2 0.9 2021-07-20 0.0207 0.0520 0.0363 0.147 0.360 0.782
4 1 2021 D3 4.58 2021-08-02 0.0251 0.0633 0.0345 0.197 0.427 0.835
5 1 2021 D4 8.75 2021-08-25 0.0332 0.0680 0.0661 0.163 0.306 0.619
6 2 2021 D0 0 2021-07-07 0.0220 0.0544 0.0451 0.133 0.279 0.667
7 2 2021 D1 0.75 2021-07-15 0.0190 0.0479 0.0292 0.117 0.268 0.761
8 2 2021 D2 3.68 2021-07-20 0.0215 0.0540 0.0383 0.153 0.373 0.776
9 2 2021 D3 12.4 2021-08-02 0.0251 0.0616 0.0356 0.184 0.393 0.811
10 2 2021 D4 20.8 2021-08-25 0.0356 0.0661 0.0759 0.132 0.214 0.481
# i 590 more rows
# i 5 more variables: ndre <dbl>, cire <dbl>, sipi <dbl>, mndvi <dbl>,
# category <chr>
```

We will change data type for the *category* column from character to factor:

```
covar21$category <- as.factor(covar21$category)
```

Let's see a summary of the dataset:

```
summary(covar21)
```

Plot	year	date	severity
Min. : 1.00	Min. :2021	Length:600	Min. : 0.000
1st Qu.:10.75	1st Qu.:2021	Class :character	1st Qu.: 0.200
Median :20.50	Median :2022	Mode :character	Median : 2.275
Mean :20.50	Mean :2022		Mean : 7.865
3rd Qu.:30.25	3rd Qu.:2023		3rd Qu.:11.875
Max. :40.00	Max. :2023		Max. :60.750

new	blue	green	red
Min. :2021-07-07	Min. :0.01718	Min. :0.04180	Min. :0.02762
1st Qu.:2021-07-15	1st Qu.:0.02097	1st Qu.:0.05603	1st Qu.:0.03330
Median :2021-07-20	Median :0.02233	Median :0.05833	Median :0.03757
Mean :2021-07-26	Mean :0.02397	Mean :0.05889	Mean :0.04288
3rd Qu.:2021-08-02	3rd Qu.:0.02522	3rd Qu.:0.06156	3rd Qu.:0.04416
Max. :2021-08-25	Max. :0.03781	Max. :0.07335	Max. :0.08514

redge	nir	ndvi	ndre
Min. :0.1147	Min. :0.1793	Min. :0.4283	Min. :0.1995
1st Qu.:0.1368	1st Qu.:0.2846	1st Qu.:0.6837	1st Qu.:0.3278
Median :0.1483	Median :0.3299	Median :0.7639	Median :0.3671

```

Mean   :0.1550   Mean   :0.3319   Mean   :0.7278   Mean   :0.3512
3rd Qu.:0.1682   3rd Qu.:0.3939   3rd Qu.:0.8114   3rd Qu.:0.3969
Max.    :0.2318   Max.    :0.4795   Max.    :0.8664   Max.    :0.4413

      cire      sipi      mndvi      category
Min.   :0.5166   Min.   :1.025   Min.   : -1.0859   0:249
1st Qu.:1.0223   1st Qu.:1.062   1st Qu.: -0.3058   1:176
Median :1.2022   Median :1.092   Median : -0.0467   2:105
Mean   :1.1493   Mean   :1.139   Mean   : -0.1076   3: 32
3rd Qu.:1.3594   3rd Qu.:1.182   3rd Qu.: 0.1316   4: 17
Max.    :1.6325   Max.    :1.475   Max.    : 0.5812   5: 13
                                6:  8

```

Now, we will filter several columns of this tibble using the *dplyr* package. First, get the column names:

```
names(covar21)
```

```

[1] "Plot"      "year"      "date"      "severity"  "new"      "blue"
[7] "green"     "red"       "redge"     "nir"       "ndvi"     "ndre"
[13] "cire"      "sipi"      "mndvi"     "category"

```

Now, the selection:

```
covar21 |> select(Plot,ndvi,ndre, cire, sipi, mndvi, category) -> covar
```

Let's check what we get:

```
covar
```

```

# A tibble: 600 × 7
  Plot ndvi ndre cire sipi mndvi category
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>
1     1 0.657 0.322 0.999  1.21 0.499  0
2     1 0.760 0.397 1.38   1.10 0.145  0
3     1 0.782 0.410 1.45   1.09 0.0765 0
4     1 0.835 0.372 1.23   1.04 -0.520  1
5     1 0.619 0.300 0.913  1.23 0.0861 1
6     2 0.667 0.339 1.07   1.20 0.438  0
7     2 0.761 0.382 1.29   1.09 0.0458 0
8     2 0.776 0.409 1.44   1.09 0.132  1
9     2 0.811 0.361 1.18   1.05 -0.458  2
10    2 0.481 0.245 0.672  1.37 0.0265 3
# i 590 more rows

```

CLS classification using machine learning (ML)

The overall process to classify the crop disease under study will be conducted using the *tidymodels* framework which is an extension of the *tidyverse* suite. It is especially focused towards providing a generalized way to define, run and optimize ML models in R. See <https://rstudio-connect.hu.nl/caait/>

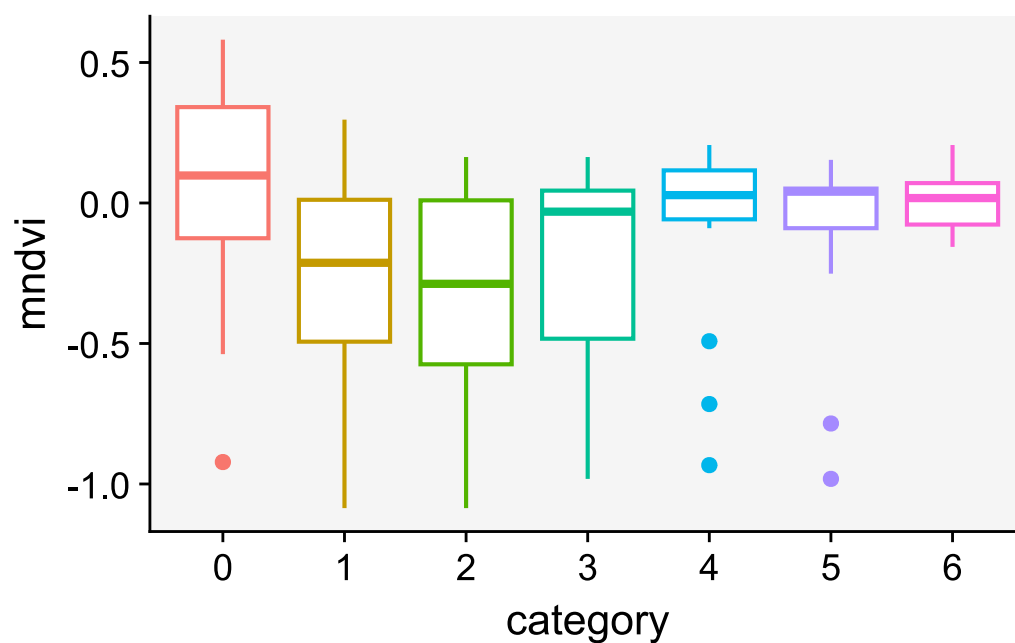
Exploratory analysis

As a first step in modeling, it's always a good idea to plot the data.

Let's try a boxplot:

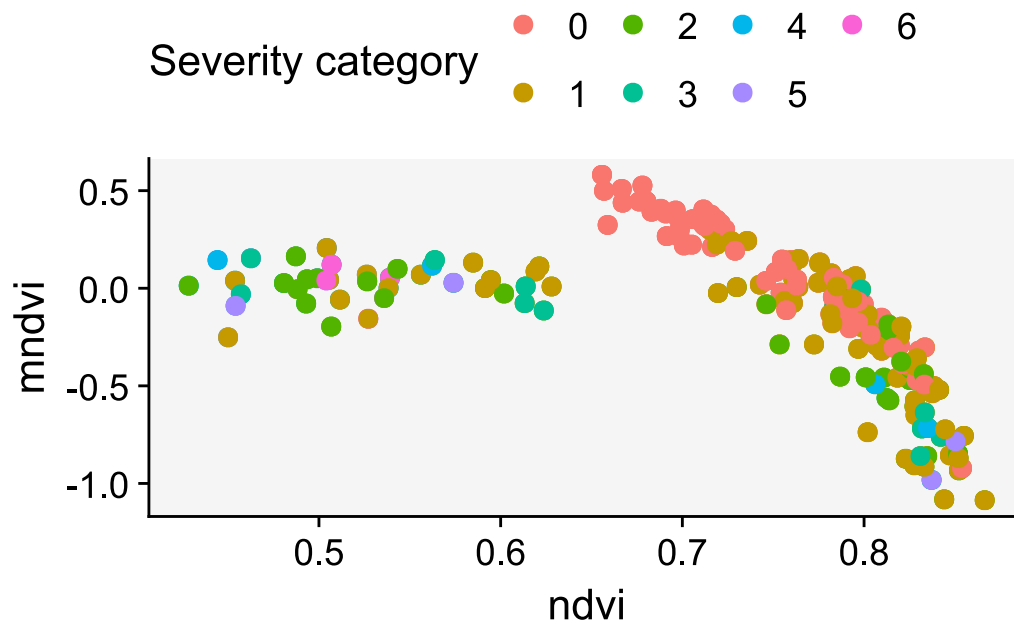
```
plot <- ggplot(covar, aes(x=category, y=mndvi, color = category))+
  geom_boxplot()+
  r4pde::theme_r4pde()+
  theme(legend.position = "none" )
```

plot



Next, we will do a scatterplot to visualize the relationship between the *ndvi* and *mndvi* indices:

```
ggplot(covar) +
  aes(x = ndvi, y = mndvi, color = category) +
  geom_point(shape = 16, size = 3) +
  labs(x = "ndvi", y = "mndvi", color = "Severity category") +
  r4pde::theme_r4pde()+
  theme(legend.position = "top")
```



Splitting the data

Next step is to divide the data into a training and a test set. The `set.seed()` function can be used for reproducibility of the computations that are dependent on random numbers. By default, the training/testing split is 0.75 to 0.25.

```
set.seed(42)
data_split <- initial_split(data = covar)
data_train <- training(data_split)
data_test <- testing(data_split)
```

Let's check the result:

```
data_train
```

```
# A tibble: 450 × 7
  Plot ndvi ndre cire sipi mndvi category
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>
1    33 0.755 0.370 1.23 1.11 0.147 0
2    25 0.729 0.356 1.14 1.14 0.192 0
3    31 0.757 0.352 1.11 1.11 -0.112 1
4    15 0.818 0.341 1.07 1.05 -0.458 1
5     6 0.809 0.420 1.49 1.07 -0.167 0
6    30 0.659 0.321 1.01 1.21 0.324 0
7    10 0.852 0.394 1.33 1.03 -0.933 4
8    26 0.813 0.417 1.47 1.07 -0.214 1
9    21 0.727 0.357 1.14 1.13 0.241 0
10     5 0.797 0.327 1.02 1.07 -0.310 2
# i 440 more rows
```

Defining the model

We will use a **gradient boosting machine** algorithm which seems a robust algorithm for different applications.

```
spec_lr <-
boost_tree(
  mode = "classification",
  trees=1000,
  #tree_depth = tune::tune(),
  #learn_rate = tune::tune(),
  #loss_reduction = tune::tune()
) |>
set_engine("xgboost", objective = "multi:softprob", eval_metric = "mlogloss",)
```

Defining the recipe

The `recipe()` function to be used here has two arguments:

- A formula. Any variable on the left-hand side of the tilde (~) is considered the model outcome (here, outcome). On the right-hand side of the tilde are the predictors. Variables may be listed by name, or you can use the dot (.) to indicate all other variables as predictors.
- The data. A recipe is associated with the data set used to create the model. This will typically be the training set, so `data = data_train` here.

```
recipe_lr <-
  recipe(category ~ ., data_train) |>
  add_role(Plot, new_role = "id") |>
  step_zv(all_predictors()) |>
  step_corr(all_predictors())
```

Model performance metrics

Next, we need to specify what we would like to see for determining the performance of the model. Different modelling algorithms have different types of metrics. Because we have a multiclass classification problem (different categories of severity), we will choose the accuracy evaluation metric here.

Combine model specification and recipe into a workflow

We will want to use our recipe across several steps as we train and test our model. We will:

- Process the recipe using the training set: This involves any estimation or calculations based on the training set. For our recipe, the training set will be used to determine which predictors should be converted to dummy variables and which predictors will have zero-variance in the training set, and should be slated for removal.
- Apply the recipe to the training set: We create the final predictor set on the training set.
- Apply the recipe to the test set: We create the final predictor set on the test set. Nothing is recomputed and no information from the test set is used here; the dummy variable and zero-variance results from the training set are applied to the test set.

To simplify this process, we can use a model workflow, which pairs a model and recipe together. This is a straightforward approach because different recipes are often needed for different models, so when a model and recipe are bundled, it becomes easier to train and test workflows. We'll use the workflows package from tidymodels to bundle our model with our recipe.

Now we are ready to set up our complete modelling workflow. This workflow contains the model specification and the recipe.

```
wf_cls_wilt <-
  workflow(
    spec = spec_lr,
    recipe_lr
  )

wf_cls_wilt
```

```
== Workflow ==
Preprocessor: Recipe
Model: boost_tree()

— Preprocessor —
2 Recipe Steps

• step_zv()
• step_corr()

— Model —
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 1000

Engine-Specific Arguments:
  objective = multi:softprob
  eval_metric = mlogloss

Computational engine: xgboost
```

Fitting the GBM regression model

Now we use the workflow previously created to fit the model on our training data. We use the training partition of the data.

```
fit_lr <- wf_cls_wilt |>
  fit(data = data_train)
```

Let's check the output:

```
fit_lr
```

```
== Workflow [trained] ==
Preprocessor: Recipe
Model: boost_tree()

— Preprocessor —
2 Recipe Steps

• step_zv()
• step_corr()

— Model —
##### xgb.Booster
call:
xgboost::xgb.train(params = list(eta = 0.3, max_depth = 6, gamma = 0,
```

```

  colsample_bytree = 1, colsample_bynode = 1, min_child_weight = 1,
  subsample = 1, num_class = 7L, objective = "multi:softprob",
  nthread = 1, eval_metric = "mlogloss"), data = x$data, nrounds = 1000,
  evals = x$watchlist, verbose = 0)
# of features: 4
# of rounds: 1000
callbacks:
  evaluation_log
evaluation_log:
  iter training_mlogloss
<num>      <num>
  1        1.1744612
  2        1.0144948
  ---      ---
  999      0.3573057
 1000      0.3573040

```

Now, we will use the fitted model to estimate CLS severity in the training data:

```

rf_training_pred <-
  predict(fit_lr, data_train) |>
  # Add the true outcome data back in
  bind_cols(data_train |>
    select(category))

```

What we got?

```
rf_training_pred
```

```

# A tibble: 450 × 2
  .pred_class category
  <fct>      <fct>
1 0          0
2 0          0
3 0          1
4 1          1
5 0          0
6 0          0
7 1          4
8 1          1
9 0          0
10 1         2
# i 440 more rows

```

Let's estimate the training accuracy:

```

rf_training_pred |> # training set predictions
  accuracy(truth = category, .pred_class) -> acc_train
acc_train

```

```

# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>       <dbl>
1 accuracy multiclass 0.784

```

The accuracy of the model on the training data should be judged against the accuracy of always predicting the most frequent class (the no-information rate), which is the relevant baseline for this seven-class problem. This basically means that the model was able to learn predictive patterns from the training data. To see if the model is able to generalise what it learned when exposed to new data, we evaluate the model on our hold-out (or so-called test data). We created a test dataset when splitting the data at the start of the modelling.

Evaluating the model on the test data

Now, we will use the fitted model to estimate health status in the testing data:

```
lr_testing_pred <-
  predict(fit_lr, data_test) |>
  bind_cols(predict(fit_lr, data_test, type = "prob")) |>
  bind_cols(data_test |> select(category))
```

What we got:

```
lr_testing_pred
```

```
# A tibble: 150 × 9
  .pred_class .pred_0 .pred_1 .pred_2 .pred_3 .pred_4 .pred_5 .pred_6
  <fct>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 1          0.00000979 0.997 0.00282 0.000150 1.40e-4 1.36e-5 5.32e-6
2 1          0.0000107 0.999 0.00107 0.0000379 1.90e-5 8.19e-5 5.31e-6
3 0          0.501      0.499 0.000140 0.0000691 6.78e-5 2.01e-5 3.20e-5
4 0          0.997      0.00326 0.0000102 0.0000132 2.00e-6 3.90e-6 1.13e-6
5 2          0.00000115 0.000244 1.000 0.000201 7.16e-7 6.45e-7 2.41e-7
6 0          0.995      0.00449 0.0000203 0.0000318 2.09e-6 8.23e-6 2.49e-6
7 1          0.00444      0.995 0.0000303 0.0000604 2.12e-5 8.48e-6 1.35e-5
8 3          0.00000112 0.00407 0.00196 0.994 5.32e-6 4.55e-6 1.84e-6
9 0          1.000      0.000154 0.0000109 0.00000796 2.92e-6 1.57e-6 1.94e-6
10 1         0.00000631 0.998 0.00167 0.000459 1.95e-4 3.61e-6 3.40e-6
# i 140 more rows
# i 1 more variable: category <fct>
```

Let's compute the testing accuracy:

```
lr_testing_pred |> # test set predictions
  accuracy(category, .pred_class)
```

```
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>    <chr>      <dbl>
1 accuracy multiclass 0.527
```

The resulting accuracy is much lower than the accuracy on the training data. This means that we would need to figure out how to improve our classification model.

```
## Let's plot the AUC-ROC
lr_testing_pred |>
  conf_mat(truth = category, .pred_class) -> cm
cm
```


	Truth						
Prediction	0	1	2	3	4	5	6
0	48	14	0	1	0	0	0
1	11	19	9	2	0	1	0
2	0	6	11	5	1	0	0
3	0	2	8	0	2	0	1
4	0	0	1	1	1	0	1
5	0	1	2	2	0	0	0
6	0	0	0	0	0	0	0

Note what are the disease categories more difficult to discriminate.

In this section we trained and tested a gradient boosting model (GBM) using five spectral indices as predictor variables (i.e. NDVI, NDRE, CIRE, SIPI, and MNDVI) obtained from imagery and field data. Accuracy metrics show that both data & model need improvement.

Some ideas for further experiments:

- Include data collected in 2022 and 2023
- Refine CLS severity categories (see (Del Ponte et al., 2019))
- Compute other vegetation indices as advised in relevant literature
- Compute other covariates (e.g. image texture metrics)

This section illustrated how to use VIs derived from UAV-based multispectral imagery and ground data to develop a classification model for estimating CLS in table beet. The results showed that a gradient boosting machine model is unable to estimate severity from several VIs with a good accuracy. We can explore the use of additional covariates to improve accuracy of the initial model.

Summary

We have completed a journey that hopefully may serve as foundation to apply remote sensing data and techniques to infer information useful for monitoring crop diseases.

Our journey consisted of three stages. In the first section we reviewed basic remote sensing concepts and provided a summary of applications of remote sensing of crop diseases. In the second section we navigated a case study on identification of banana Fusarium wilt from multispectral UAV imagery. In the third section we moved to a case study dealing with estimation of cercospora leaf spot disease severity on table beet. For the two case studies we used datasets publicly available.

Now, it is time to conclude with several reflections about: (i) what we learned in this chapter; (ii) what are the potential and limitations of remote sensing for identification & monitoring crop diseases; and (iii) what are main challenges to overcome to advance the use of remote sensing techniques for identification and quantification of crop diseases.

Lessons learned in this journey

First of all, we started to become familiar with concepts and techniques that can guide us in future studies focused on the use of remote sensing for crop disease monitoring. Although they are only the tip of the remote sensing iceberg, which we must know in depth, they represent part of the foundation necessary to be successful in any research related to the topic of this chapter.

Along the way, we learned to use some R libraries that allow us to: (i) read, visualize and process multispectral images and geospatial data; (ii) merge raster and vector data and extract spectral covariates related to plant biophysical, biochemical, and functional traits; and (iii) apply machine learning algorithms to infer information useful for plant disease identification and quantification.

The results obtained in our hands-on activities suggest that the application of machine learning algorithms to learning from data is driven by an iterative process: starting from an initial workflow, users need to iteratively modify their workflow, based on previous results, to improve performance.

We have learned that, in order to use data properly, research data needs to be fully described and documented. Otherwise, users may become frustrated or lost in guessing, cleaning, and pre-processing tasks. We have also learned that if the code is well documented, it is feasible to reuse it in future studies.

Potential and limitations of remote sensing for crop disease monitoring

A variety of remote sensing (RS) devices can be used to obtain data on the spectral response of plants and crops to diseases and insect damage. Processing and analyzing such data allows to estimate information useful for monitoring crops and analysing different stages of plant health. As individual devices have some limitations, multimodal remote sensing data & techniques (i.e. reflectance, fluorescence, thermal, microwave, lidar, multiangular sensing) are increasingly being used in research.

In the multimodal setting, data from different platforms & instruments are used together to obtain a better outcome. Multimodal methods allow for the integration of spectral responses from different & complementary electromagnetic radiation ranges, allowing to quantify different plant responses (e.g. stomatal closure, increased respiration, decreased chlorophyll, altered pigments, modified biochemistry, photosynthetic inhibition, altered growth, modified leaf angle, water content), and analyzing them to obtain a comprehensive understanding of disease progression, including eventual recovery. By using multimodal methods, it is possible to obtain a more accurate and detailed analysis of crop conditions, leading to advanced knowledge, improved decision-making and ultimately, better crop health management.

Main challenges to overcome

Building more resilient agricultural systems requires concurrent and near-real-time and reliable crop information for effective decision making. Using remote sensing data & techniques for monitoring crop health can become a relevant method to derive crop information at leaf, canopy, field, and farm scales.

In spite of the increasing literature reporting the application of remote sensing for monitoring crop diseases, it seems that there is a lack of open, transparent and reproducible data and methods to ensure a robust evaluation of performance, reliability and generalization of machine learning and deep learning models, a necessary pre-requisite to safely advance the applicability of remote sensing for crop monitoring in the field.

To fully realize the potential of remote sensing for estimating crop health status, there is a need of reliable ground truth data to enable both data calibration (i.e. linking in-situ to remote values) and accuracy assessment of estimations. Since field measurements require a lot of work and time, there is always a risk that the sampling is not correct (e.g. not enough samples, incorrect sample location, or the sampled area does not match the sensor footprint).

Both collaborative research and crowdsourcing could become a solution to overcome the restrictions of ground-truth data accessibility as well as the lack of open, transparent, and reusable remote sensing data & workflows. It is anticipated that joint multidisciplinary research as well as user participation in the complete process of crop monitoring from remote sensing could improve the reliability of crop health information.

PART II

Temporal analysis

9. Disease progress curves

How epidemics occur

Before knowing how epidemics develop in time, it is important to understand how an epidemic occurs. 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 encounters 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 measured in time units. Finally, the **infectious period** takes place and continues until the pathogen is no longer capable of producing the **secondary inoculum** on the infected site.

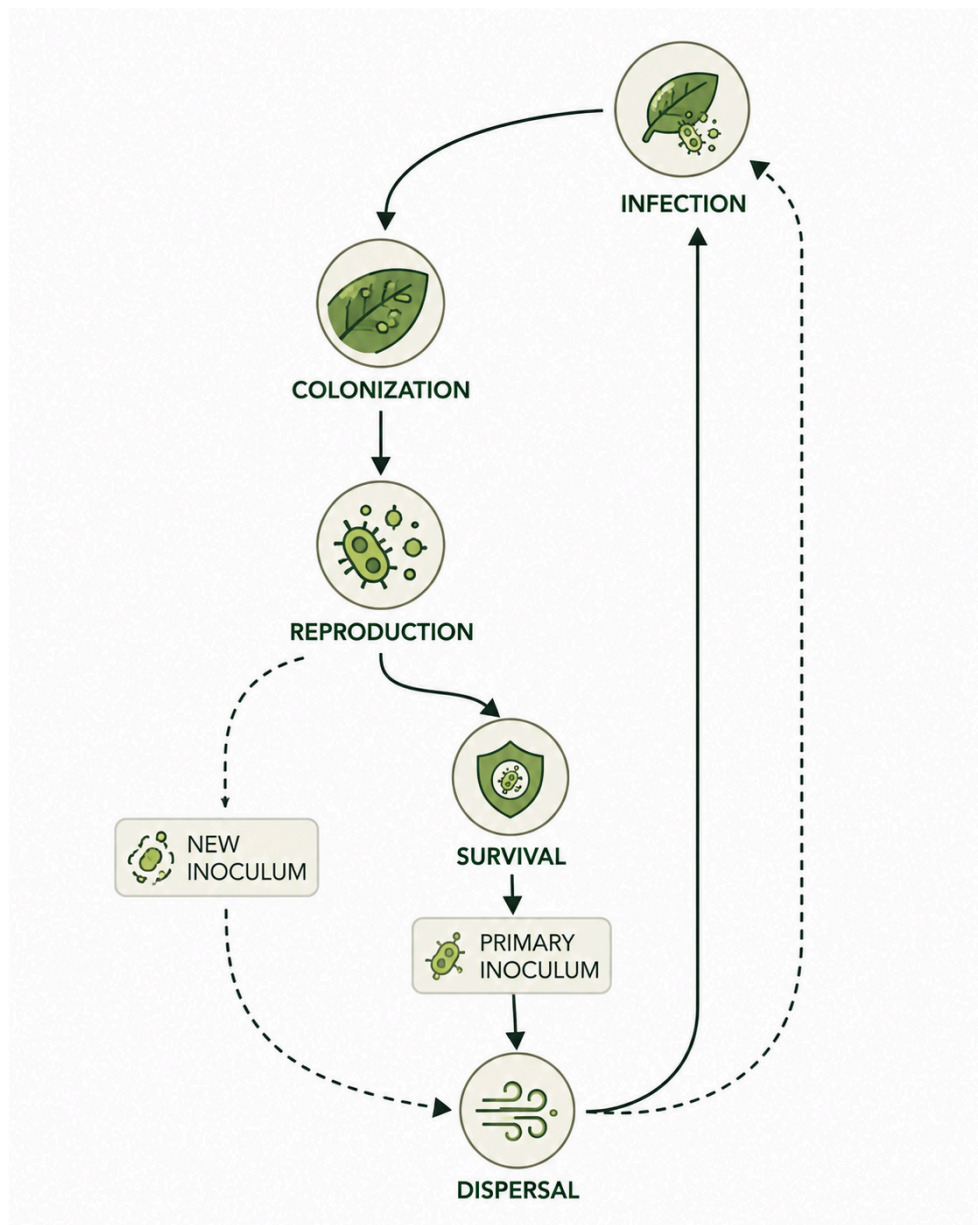


Figure 9.1: Five main processes of the disease cycle

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."

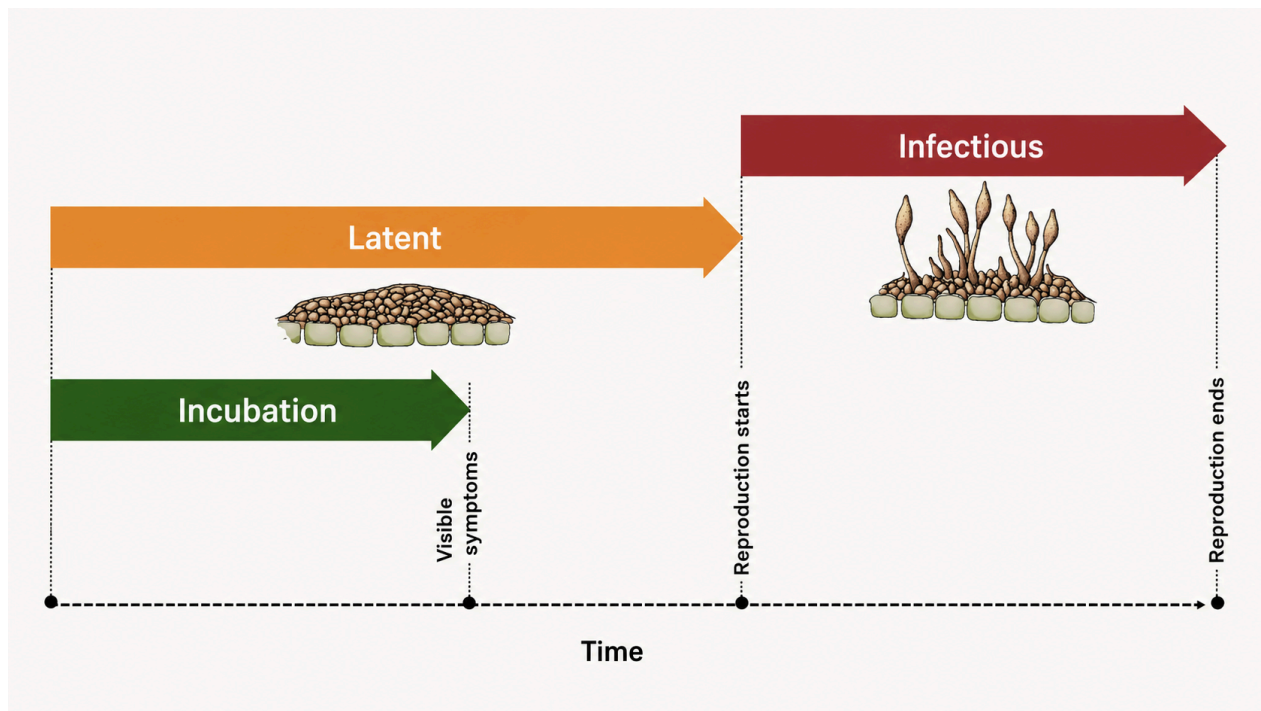


Figure 9.2: Three time-related epidemiological periods and their relations with stages of the disease cycle including colonization (symptoms) and reproduction (sporulation in the case of fungi). Drawings of apple scab symptoms and signs adapted from Agrios (2005)

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 over time. 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
library(r4pde)
theme_set(theme_r4pde()) # set global theme
```

There are several ways to create a data frame in R. We use the `tribble` function as below. The entered data will be assigned to a data frame 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(linewidth = 1)+
  geom_point(size = 3, shape = 16)+
  labs(x = "Assessment time (days)",
       y = "Disease intensity (%)")

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

```

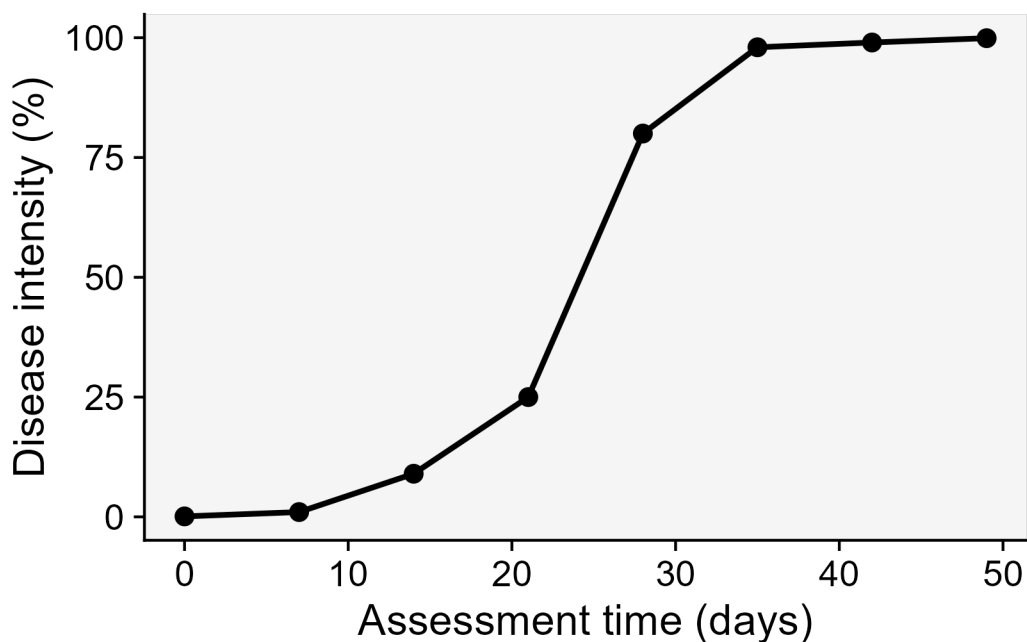


Figure 9.3: A typical disease progress curve for an epidemic that reaches the maximum value

Epidemic classification

Vanderplank analyzed the shapes of a 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. Typically, the progress curves for monocyclic epidemics have a saturation type shape. An example of a monocyclic epidemic is the disease known as wheat smut caused by

Tilletia caries. In this disease, although two types of spores are produced during a single cycle (teliospores and basidiospores), no additional infection cycles occur within the same growing season.

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 9.4). An example of a polycyclic epidemic is soybean rust as it undergoes repeated infection cycles with urediniospores on soybean.

```
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)
```

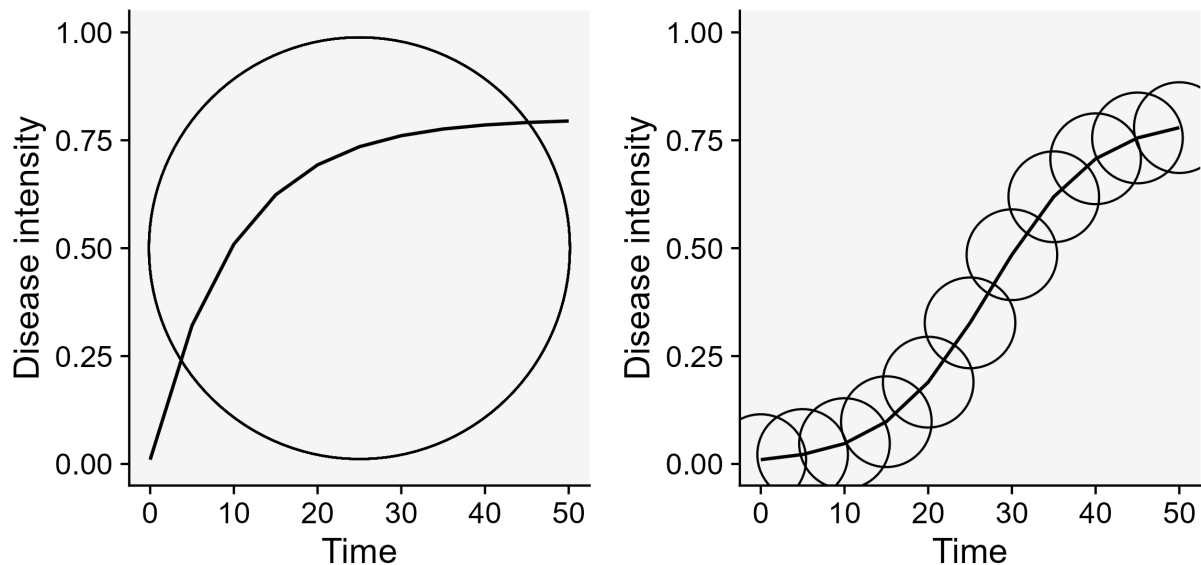


Figure 9.4: Hypothetical curves for monocyclic (left) and polycyclic (right) epidemics. Each circle represents a single infection cycle.

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 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")
```

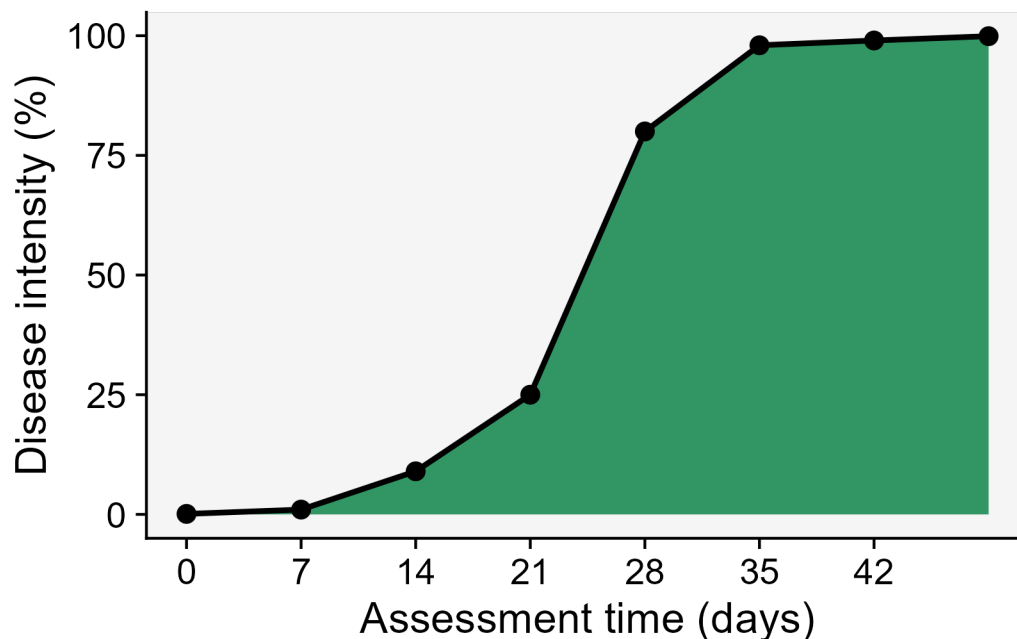


Figure 9.5: Representation of the area under the disease progress curve

The AUDPC summarizes the “total measure of disease stress” and is largely used to compare epidemics (Jeger & 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")

```

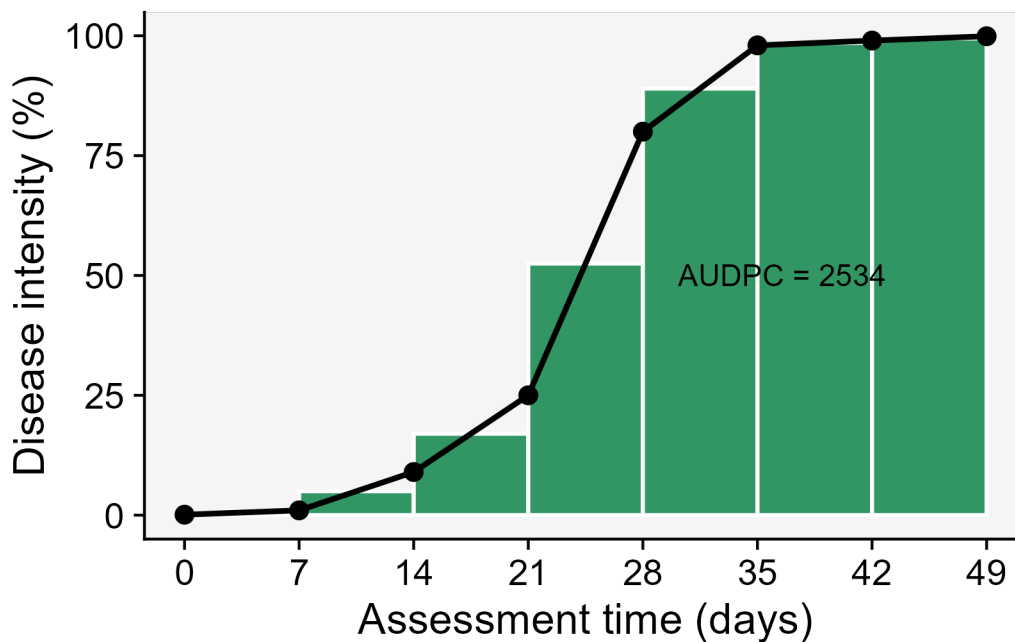


Figure 9.6: Representation of the area under the disease progress curve calculated using the trapezoidal method

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 & Piepho, 2012).

```

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
```

Functional comparison of curves

The AUDPC is useful when the objective is to reduce an epidemic to one number. However, two epidemics may have similar AUDPC values and still differ in timing, shape, or the period when disease increases most rapidly. A more direct way to compare whole disease progress curves is to treat each curve as a function of time. This is the idea behind the `functional_curves()` and `functional_distances()` functions in the *r4pde* package.

The `functional_curves()` function fits a smooth disease progress curve for each treatment using a generalized additive model (GAM). The fitted curves are then predicted over a common time grid. The `functional_distances()` function compares these predicted functions by calculating the distance between each pair of curves across the entire time interval. For two fitted curves, $f_1(t)$ and $f_2(t)$, the distance is based on the integrated squared difference between them:

$$d(f_1, f_2) = \sqrt{\int \{f_1(t) - f_2(t)\}^2 dt} \quad (9.1)$$

Because disease progress data are observed at discrete times, the function uses predictions on a regular grid and approximates the integral numerically. In practical terms, the distance increases when two curves are far apart for a long period of time, and decreases when the curves remain close to each other.

The example below creates two hypothetical epidemics with nearly identical AUDPC values but very different shapes. The first epidemic increases earlier and reaches a moderate plateau. The second epidemic remains low for a longer period but then increases sharply and reaches a higher final intensity. If only the AUDPC is used, these two epidemics appear almost equivalent; if the full curves are compared, their temporal patterns are clearly different.

```
dpc_functional <- tidyr::crossing(
  treatment = c("Early plateau", "Late surge"),
  unit_id = 1:4,
  t = seq(0, 49, 7)
) |>
mutate(
  y = case_when(
    treatment == "Early plateau" ~ 0.70 * plogis(-3 + 0.24 * t),
    treatment == "Late surge" ~ plogis(-8.5 + 0.36 * t)
  )
)

audpc_functional <- dpc_functional |>
distinct(treatment, t, y) |>
group_by(treatment) |>
summarise(
  audpc = epifitter::AUDPC(t, y, y_proportion = TRUE),
  .groups = "drop"
```

```
)

curves_fit <- functional_curves(
  data = dpc_functional,
  time = "t",
  response = "y",
  treatment = "treatment",
  response_scale = "proportion",
  min_points = 5,
  grid_n = 120,
  family_try = "quasibinomial",
  show_progress = FALSE
)

audpc_functional
```

```
# A tibble: 2 × 2
  treatment    audpc
  <chr>      <dbl>
1 Early plateau 25.4
2 Late surge   25.4
```

```
curve_dist <- functional_distances(
  curves_fit,
  cluster_k = 2,
  show_progress = FALSE
)

curve_dist$distance_table
```

```
      group1    group2 distance
1 Early plateau Late surge 1.905074
```

The figure first shows that the two epidemics have the same AUDPC even though their shapes differ. The lower panels show how the functional distance is built. The two fitted curves are compared at every point in the prediction grid. The vertical segments represent local differences between the curves. The shaded region emphasizes the separation accumulated over time. The final distance is the square root of the summed squared differences, scaled by the spacing of the time grid.

```
distance_curves <- curves_fit$curves |>
  tidyr::pivot_wider(
    names_from = treatment,
    values_from = mu
  ) |>
  mutate(
    ymin = pmin(`Early plateau`, `Late surge`),
    ymax = pmax(`Early plateau`, `Late surge`),
    squared_difference = (`Early plateau` - `Late surge`)^2
  )

distance_lines <- distance_curves |>
  slice(round(seq(1, n(), length.out = 12)))
```

```

distance_value <- curve_dist$distance_table$distance[1]

audpc_labels <- audpc_functional |>
  mutate(
    t = 26,
    y = 0.08,
    label = paste0("AUDPC = ", round(audpc, 2))
  )

p_same_audpc <- dpc_functional |>
  distinct(treatment, t, y) |>
  ggplot(aes(t, y, color = treatment, fill = treatment)) +
  geom_area(alpha = 0.22, color = NA) +
  geom_line(linewidth = 1) +
  geom_point(size = 2) +
  geom_label(
    data = audpc_labels,
    aes(t, y, label = label),
    inherit.aes = FALSE,
    size = 3.2,
    fill = "white"
  ) +
  facet_wrap(~ treatment, nrow = 1) +
  scale_y_continuous(labels = scales::percent, limits = c(0, 1)) +
  labs(
    x = "Assessment time (days)",
    y = "Disease intensity"
  ) +
  theme_r4pde(font_size = 12) +
  theme(
    legend.position = "none",
    strip.background = element_blank(),
    strip.text = element_text(face = "bold")
  )

p_distance_curves <- curves_fit$curves |>
  ggplot(aes(t, mu, color = treatment)) +
  geom_ribbon(
    data = distance_curves,
    aes(x = t, ymin = ymin, ymax = ymax),
    inherit.aes = FALSE,
    fill = "#D9A441",
    alpha = 0.30
  ) +
  geom_segment(
    data = distance_lines,
    aes(x = t, xend = t, y = ymin, yend = ymax),
    inherit.aes = FALSE,
    color = "gray35",
    linewidth = 0.5
  ) +
  geom_point(
    data = dpc_functional,
    aes(t, y, color = treatment),
    alpha = 0.45,
    size = 1.8
  ) +
  geom_line(linewidth = 1.1) +

```

```

annotate(
  "label",
  x = 34,
  y = 0.16,
  label = paste0("Functional distance = ", round(distance_value, 2)),
  size = 3.5,
  fill = "white"
) +
scale_y_continuous(labels = scales::percent, limits = c(0, 1)) +
labs(
  x = "Assessment time (days)",
  y = "Disease intensity",
  color = "Treatment"
) +
theme_r4pde(font_size = 12) +
theme(legend.position = "bottom")

p_squared_difference <- distance_curves |>
ggplot(aes(t, squared_difference)) +
geom_area(fill = "#339966", alpha = 0.65) +
geom_line(color = "gray60", linewidth = 0.8) +
labs(
  x = "Assessment time (days)",
  y = "Squared difference"
) +
theme_r4pde(font_size = 12)

library(patchwork)
functional_distance_plot <- p_same_audpc / p_distance_curves / p_squared_difference +
  plot_layout(heights = c(1.2, 1.6, 1))

ggsave(
  "imgs/functional-distance.png",
  functional_distance_plot,
  width = 7,
  height = 8,
  bg = "white"
)

```

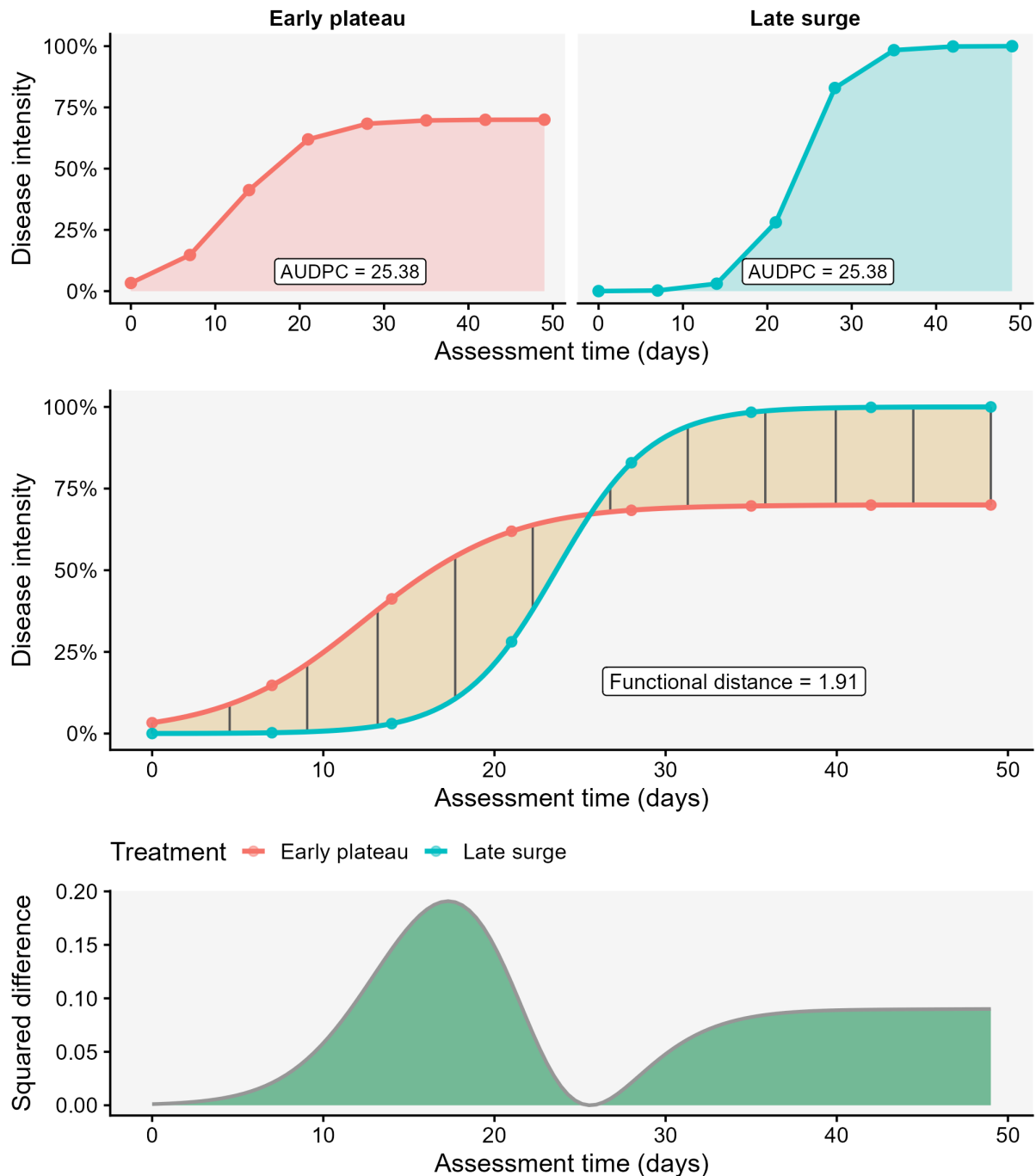



Figure 9.7: Two disease progress curves with the same AUDPC but different temporal shapes. The upper panels show equal AUDPC values. The middle panel shows the fitted functional curves and the vertical differences between them at selected times. The lower panel shows the squared differences accumulated across the time grid.

The same approach can be extended to many treatments or genotypes. In that case, `functional_distances()` returns a full distance matrix, a pairwise distance table, and clusters of curves with similar epidemic patterns. These outputs are useful when the interest is not only whether one treatment has less disease, but also whether the entire temporal pattern of the epidemic is different.

10. 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., 2007b). 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.

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 is at least one additional (third) parameter.

Following the convention proposed by Madden et al. (2007b) 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.

Exponential

The differential equation for the exponential model is given by

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

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 linearly with the increase in y (so that y itself grows exponentially over time).

The integral for the exponential model is given by

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

where y_0 and r_E are obtained via estimation. Let's simulate two curves by varying r while fixing y_0 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")
```

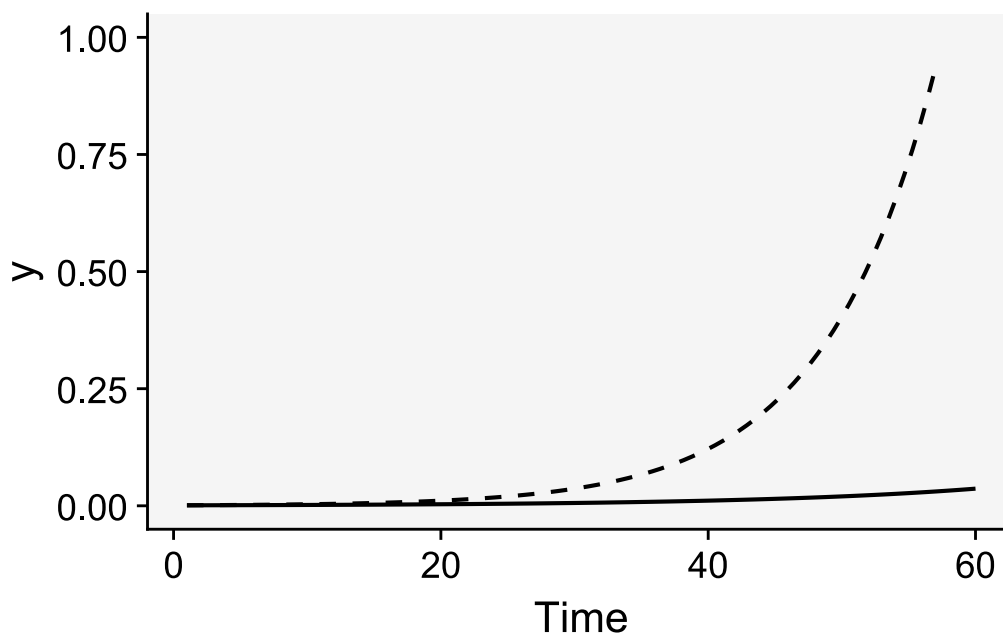


Figure 10.1: Exponential curves with two rates of infection (0.06 and 0.12) and the same initial inoculum (0.001)

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

```
dat |>
  ggplot(aes(t, y)) +
    stat_function(fun = function(t) y0 * exp(r * 2 * t), linetype = 1) +
    stat_function(fun = function(t) y0 * 5 * exp(r * 2 * t), linetype = 2) +
    ylim(0, 1) +
    theme_r4pde()+
    labs(x = "Time")
```

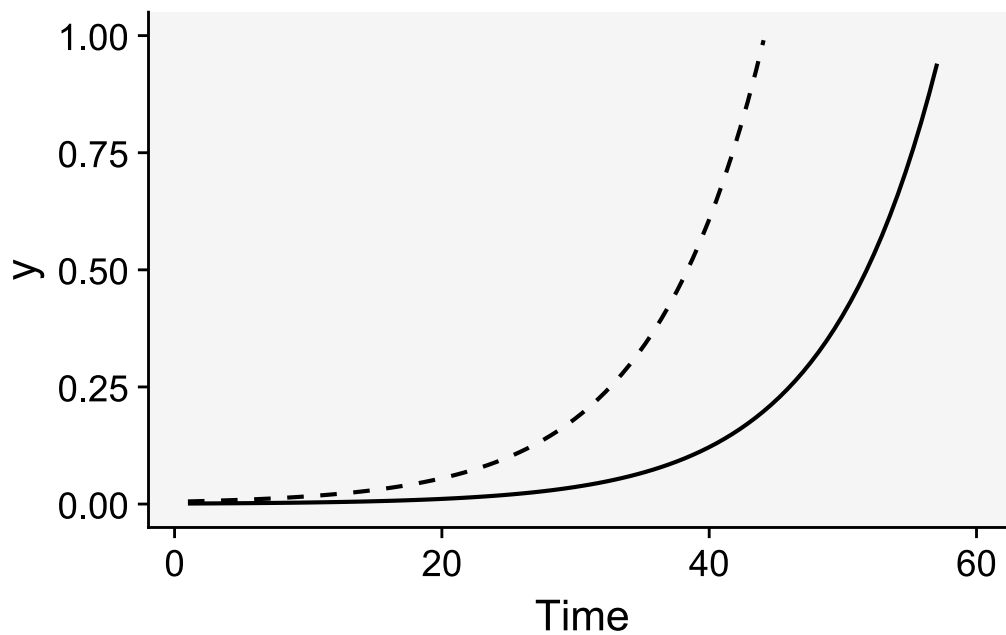


Figure 10.2: Exponential curves with the same rate of infection (0.12) and single and five times the initial inoculum (0.001)

Monomolecular

The differential of the monomolecular model is given by

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

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

The integral of the monomolecular model is given by

$$y = 1 - (1 - y_0)e^{-r_M t}$$

This model commonly describes the temporal patterns of the monocyclic epidemics. In those, the inoculum produced during the course of the epidemics does 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 curves 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")
```

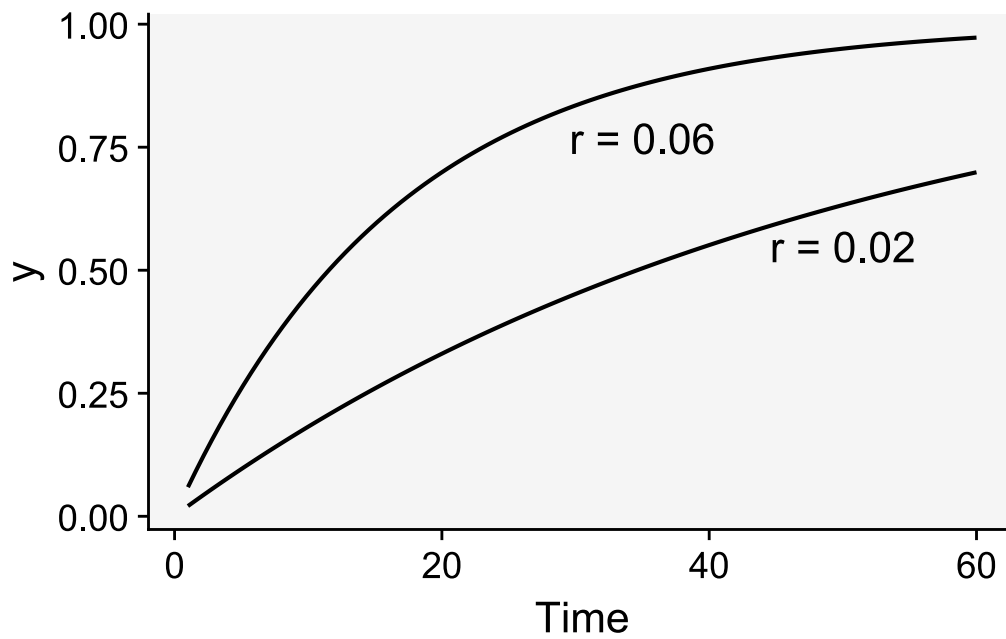


Figure 10.3: Monomolecular curves with two rates of infection (0.06 and 0.02) and the same initial inoculum (0.001)

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")
```

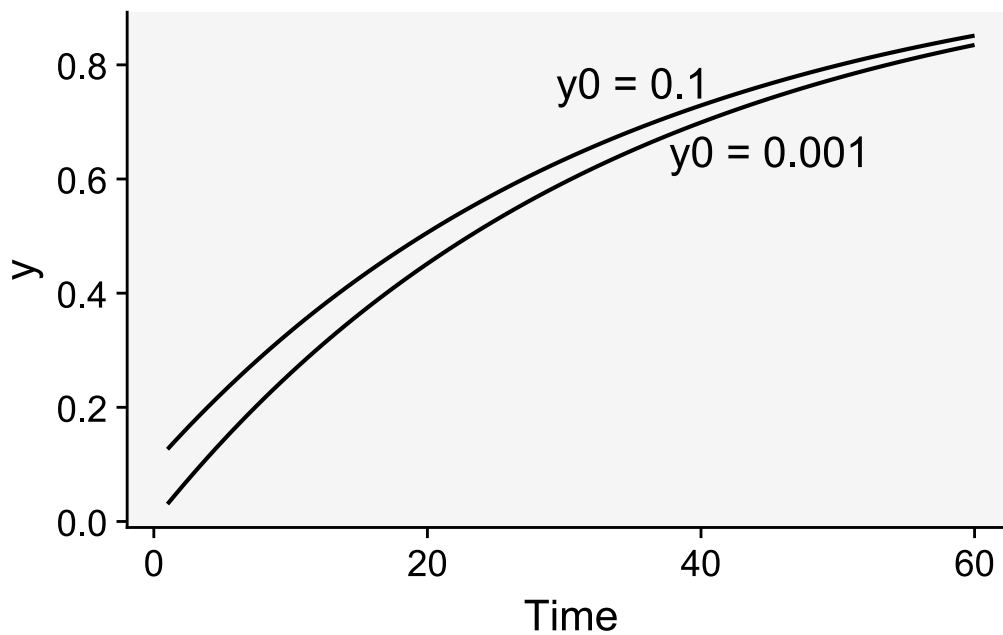


Figure 10.4: Monomolecular curves with one rate (0.03) and the initial inoculum increased 100 times

Logistic

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

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

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

Biologically, y in its differential equation implies that $\frac{dy}{dt}$ increases with the increase in y (as in the exponential) because more disease means more inoculum. However, $(1 - y)$ leads to a decrease in $\frac{dy}{dt}$ when y approaches the maximum $y = 1$, because the proportion of healthy individuals or host area decreases (as in the monomolecular). Therefore, $\frac{dy}{dt}$ is minimal at the onset of the epidemics, reaches a maximum when $y = 0.5$ and declines until $y = 1$.

The integral is given by

$$y = \frac{1}{1 + \frac{1-y_0}{y_0} e^{-r_L t}},$$

where r_L is the apparent infection rate of the logistic model and y_0 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 = 0.24") +
  annotate(geom = "text", x = 50, y = 0.10, label = "r = 0.12")
```

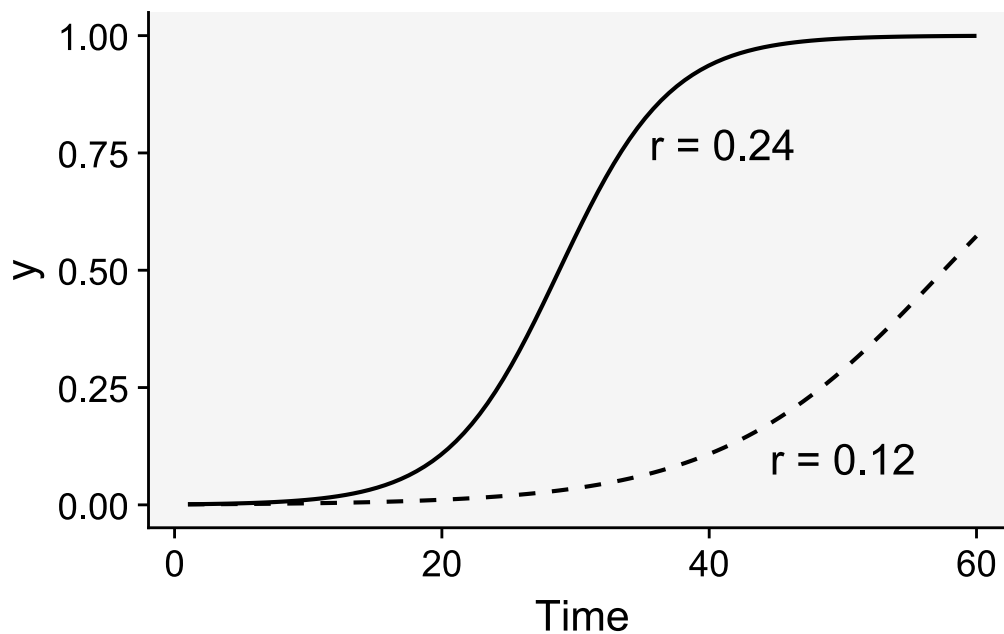


Figure 10.5: Logistic curves with two rates of infection (0.12 and 0.24) and the same initial inoculum (0.001)

Now the inoculum is reduced 10 times for the 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")
```

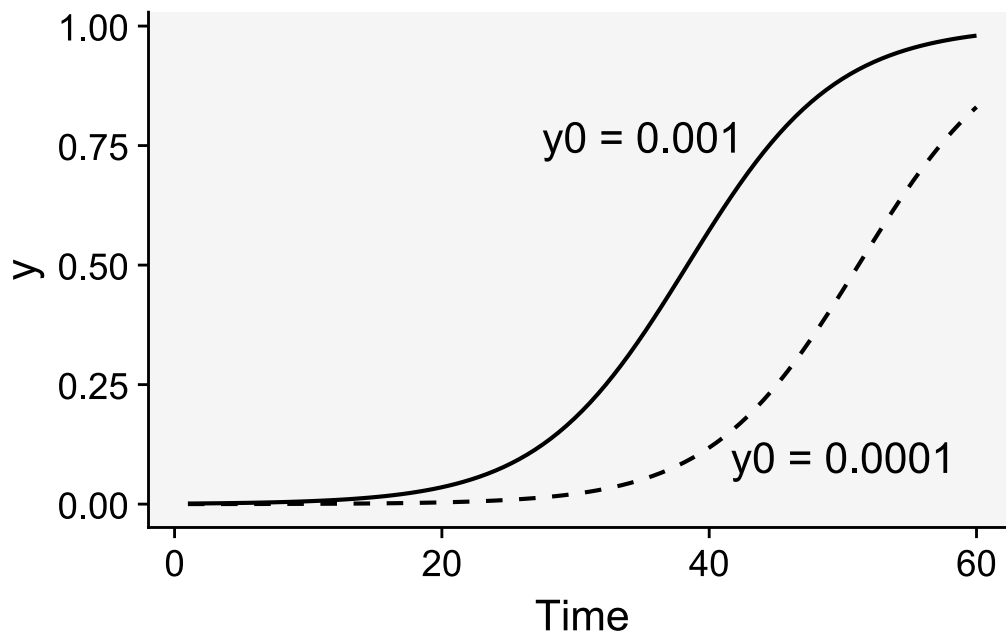


Figure 10.6: Logistic curves with a single rate of infection (0.18) and two initial inocula (0.001 and 0.0001)

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

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

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

$$y = e^{\ln(y_0) e^{-r_G t}},$$

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")
```

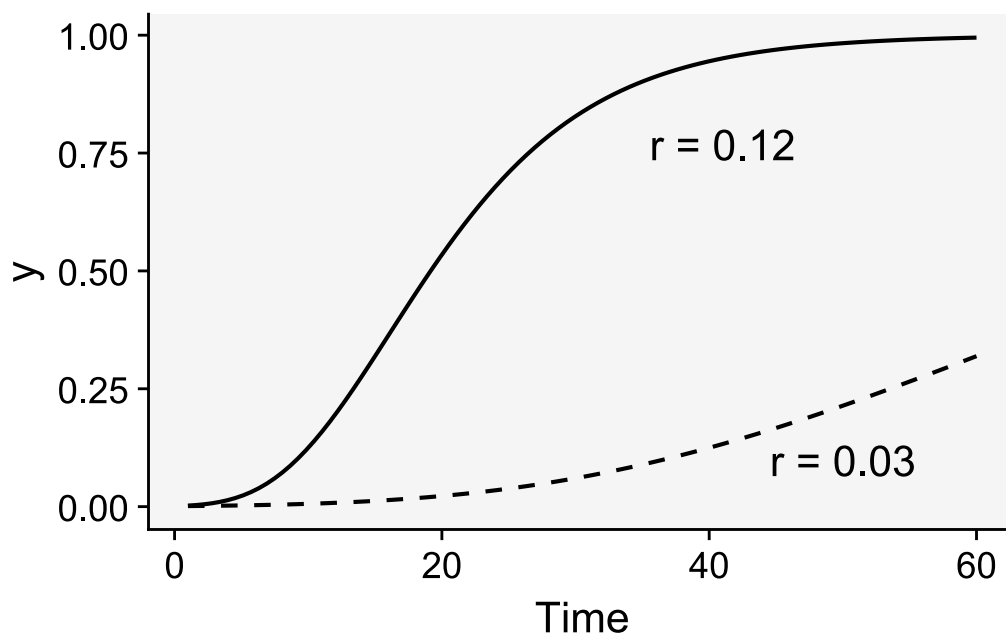



Figure 10.7: Gompertz curves with two rates of infection (0.12 and 0.03) and the same initial inoculum (0.001)

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 = 0.000001")
```

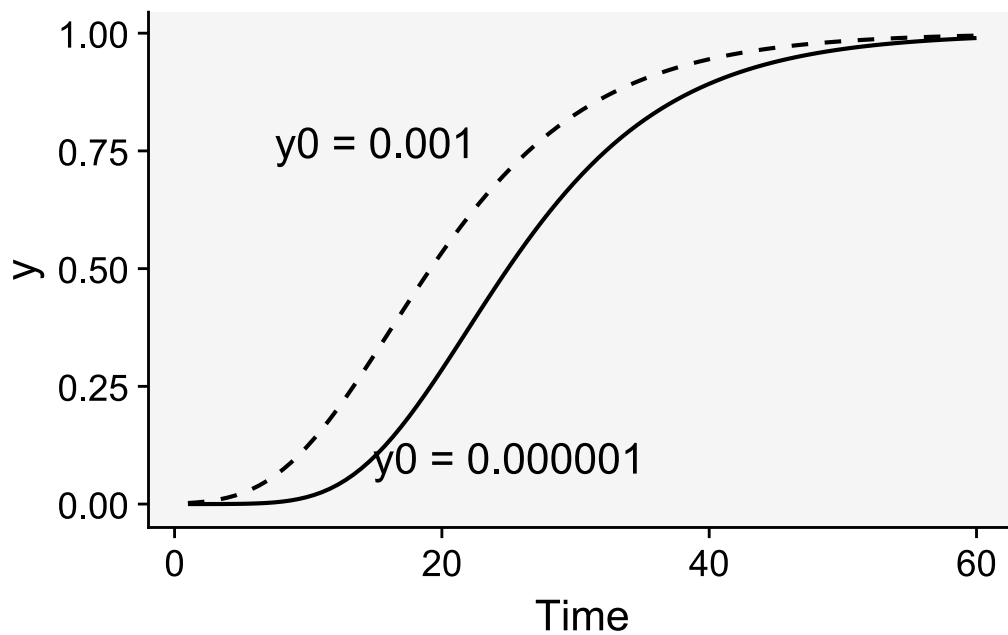


Figure 10.8: Gompertz curves with a single rate of infection (0.12) and two levels of initial inoculum (0.001 and 0.000001)

Interactive application

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

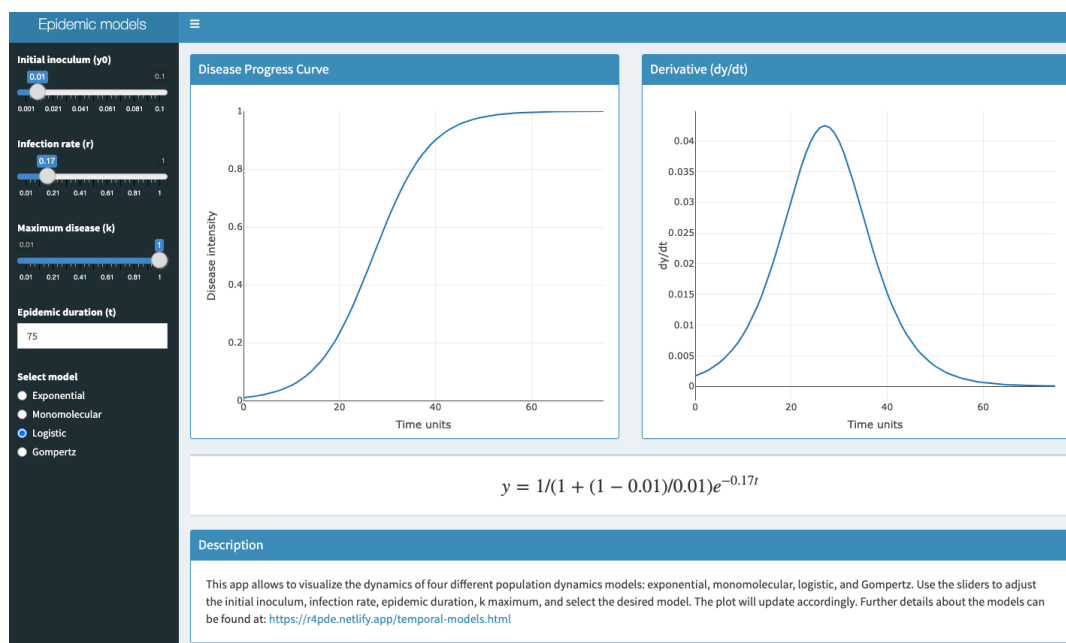


Figure 10.9: Screenshot of the application to visualize the population dynamics models by varying the model's parameters

11. 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. We'll illustrate the procedure using both built-in R functions and custom functions from the `epifitter` package (Alves & 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., 2007b). 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.

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 (%)")
```

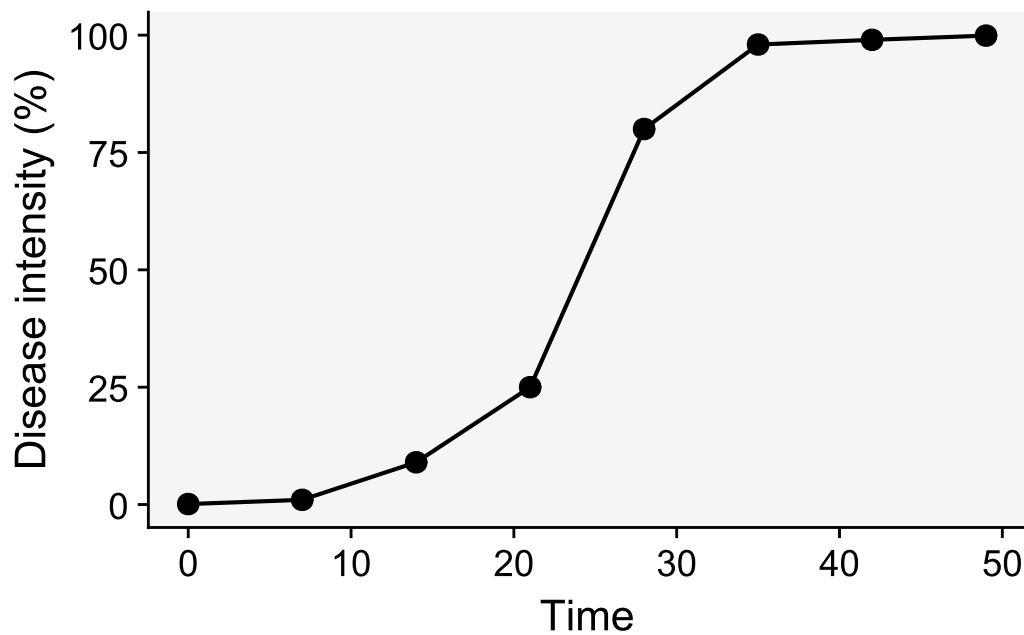


Figure 11.1: Disease progress curves for one tobacco etch epidemic in pepper. Reproduced from Madden et al. (2007b) page 94

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 y_2 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")
```

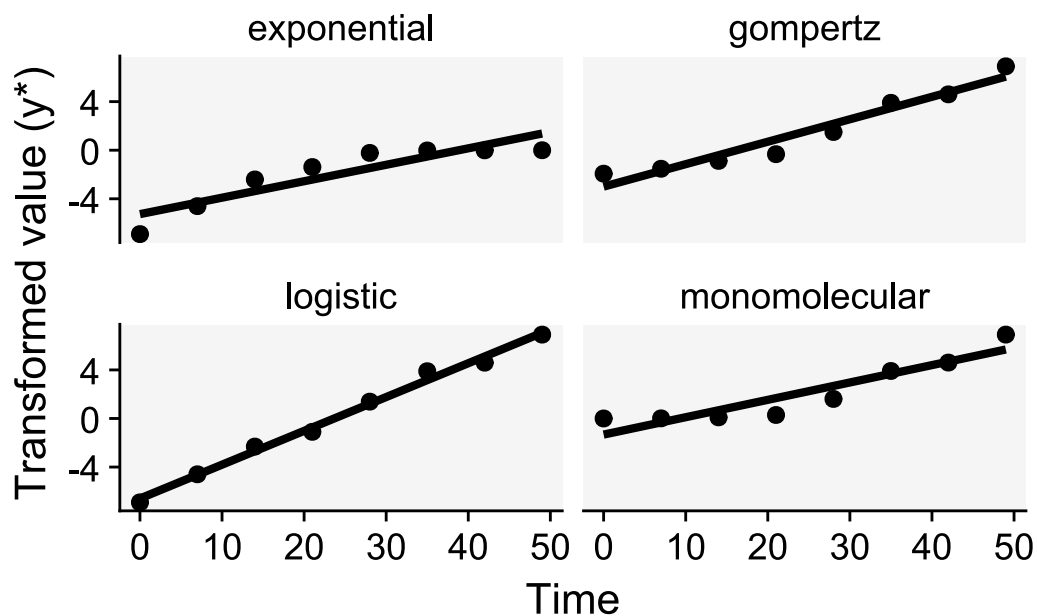


Figure 11.2: Curves of the transformed data for each epidemiological model against time. The goal is to check which of the models provides the best fit based on the straight line

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-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
```

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

```
[1] 0.4523616
```

```
# 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.

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

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

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

```
[1] 0.9431066
```

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

```
[1] 0.8407922
```

```
# 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	$\log(y)$	$\exp(y \cdot E)$
Monomolecular	$\log(1 / (1 - y))$	$1 - \exp(-y \cdot M)$
Logistic	$\log(y / (1 - y))$	$1 / (1 + \exp(-y \cdot L))$
Gompertz	$-\log(-\log(y))$	$\exp(-\exp(-y \cdot G))$

```
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")
```

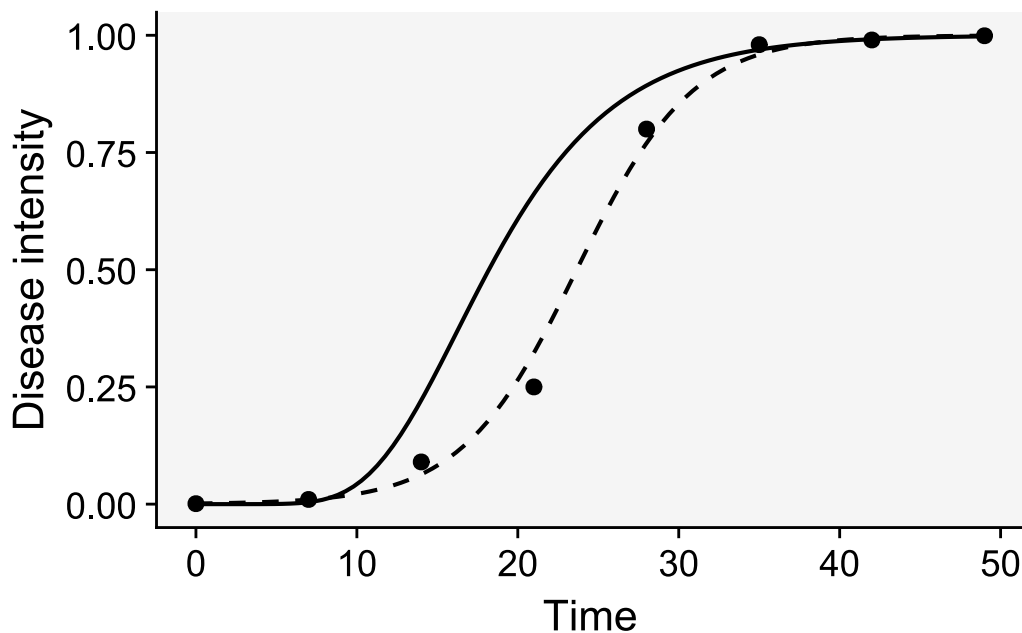


Figure 11.3: Disease progress curve and the fit of the logistic (dashed line) and the Gompertz (solid line) based on parameters estimated using linear regression

In this case, it's clear that the logistic model (the dashed 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.

Nonlinear 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 1.647e-14  3.712e-13   0.044  0.96605
r  1.626e-01  3.017e-02   5.391  0.00168 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.06782 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.

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., 2007b). 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.

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
  )
```

Visualize the DPCs

Before proceeding with model selection and fitting, let’s visualize the three epidemics. The code below closely reproduces the top plot of Fig. 4.15 (Madden et al. (2007b) 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")
```

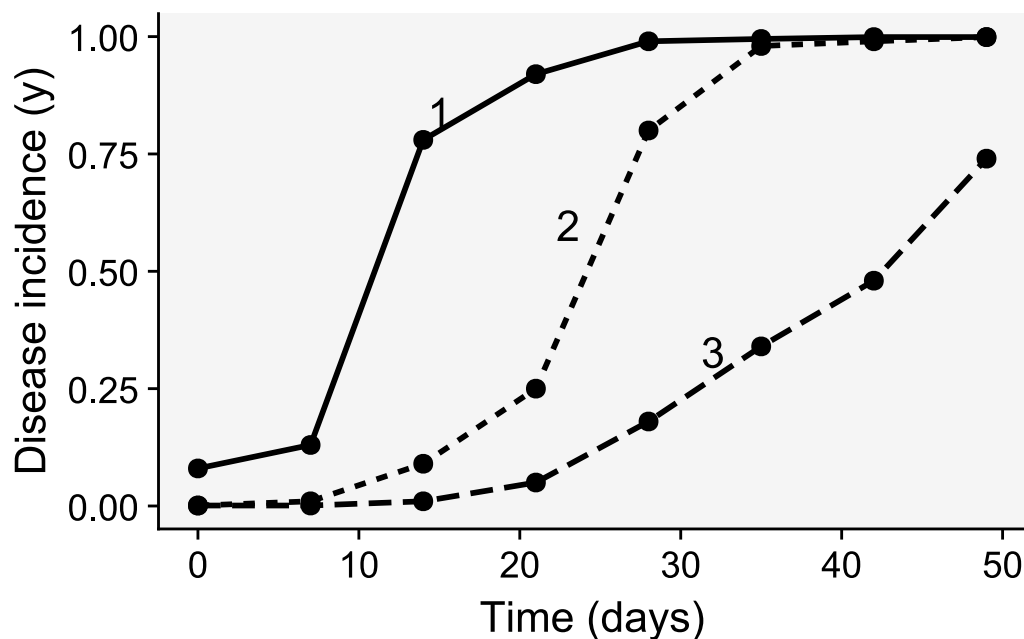


Figure 11.4: Disease progress curves for three tobacco etch epidemics in pepper. Reproduced from Madden et al. (2007b) page 94

Two 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.

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 epidemic 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 used 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
Exponential	0.7839	0.6447	0.6705
Gompertz	0.9848	0.9700	0.5911
Logistic	0.9782	0.9572	0.8236
Monomolecular	0.9838	0.9681	0.5432

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

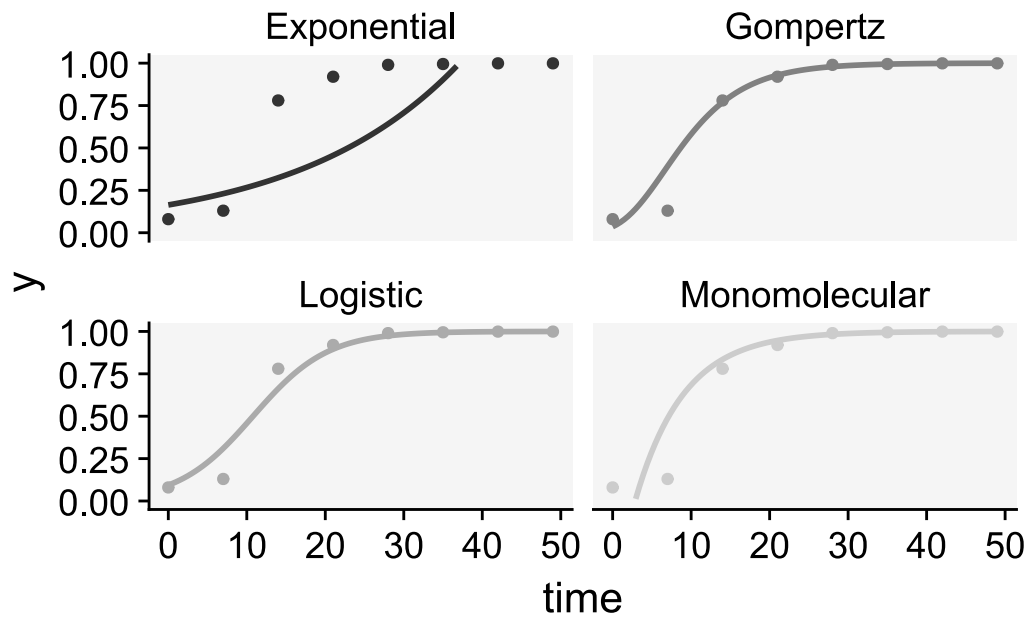
	CCC	r_squared	RSE
Exponential	0.8971	0.8134	1.2016
Gompertz	0.9707	0.9431	0.8408
Logistic	0.9962	0.9924	0.4524
Monomolecular	0.9248	0.8601	1.0684

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

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

The statistics of the model fit confirm 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.

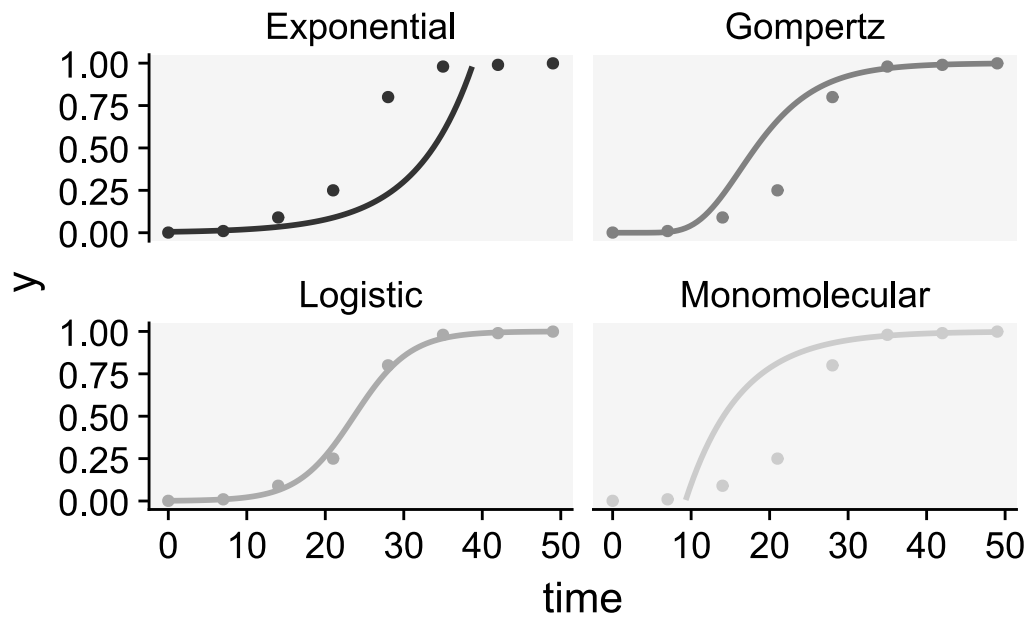
```
plot_fit(epi1)+
  ylim(0,1)+
  scale_color_grey()+
  theme_r4pde()+
  theme(legend.position = "none")
```



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

best_CCC	best_RSE	model	r	r_se	ci_lwr	ci_upr	v0	v0_rse	r_squared	RSE	CCC	y0	ci_lwr	ci_upr	rho
4	3	Exponential	0.487601	0.147802	0.125904	0.492930	0.906043	0.281634	0.656705	0.083393	0.381638	0.056804	0.172362	0.081272	
1	2	Gompertz	0.127033	0.099682	0.234542	0.503638	0.156700	0.273911	0.584785	0.355470	0.002026	0.193304	0.153928		
3	4	Logistic	0.104047	0.181546	0.598254	0.822071	0.585316	0.857248	0.235098	0.815309	0.350382	0.273207	0.747287	0.50675	
2	1	Monomolecular	0.166901	0.119739	0.290940	0.462524	0.506329	0.681054	0.319738	0.048880	0.287452	0.352661	0.08207		

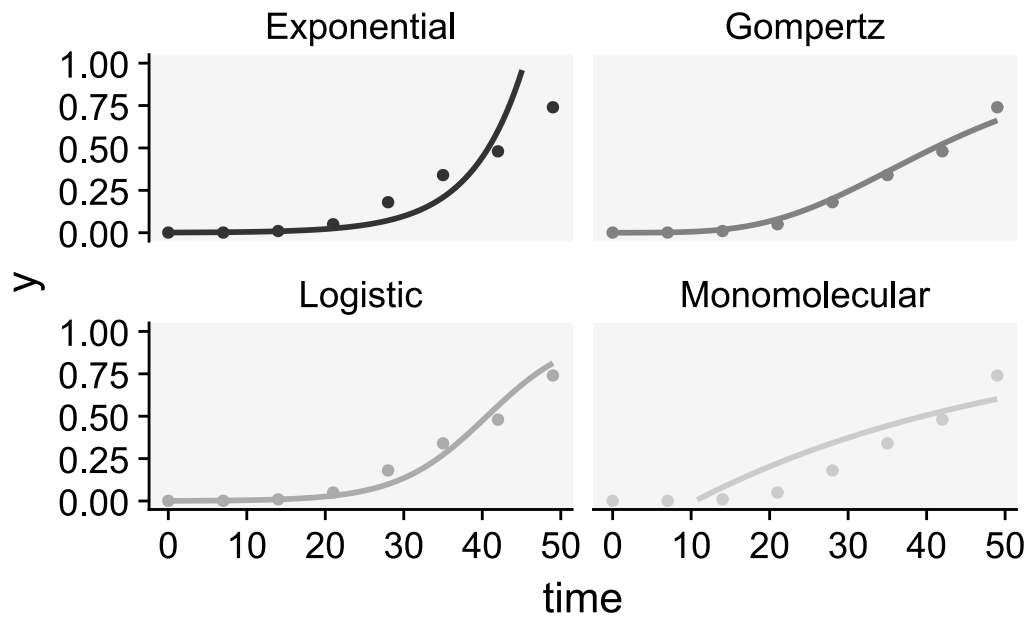
```
plot_fit(epi2)+
  ylim(0,1)+
  scale_color_grey()+
  scale_color_grey()+
  theme_r4pde()+
  theme(legend.position = "none")
```



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

best_CCC	best_RSE	model	r	r_se	ci_lwr	ci_upr	v0	v0_se	r_squared	RSE	CCC	y0	ci_lwr	ci_upr	rho
4	4	Exponential	0.954579	0.067064	0.890268	0.926391	177561.7	113401.2	0.158039	0.097100	0.905176	0.007673	0.45252	0.25763	
2	2	Gompertz	0.948373	0.085338	0.894323	0.901826	998024	2709431	0.040792	0.072000	0.900000	0.000000	0.004930	0.62095	
1	1	Logistic	0.978481	0.099725	0.838288	0.958956	2919989	2365452	0.361630	0.013702	0.800672	0.242800	0.746413		
3	3	Monomolecular	0.943026	0.035085	0.929064	0.925648	9628600	1898363	0.324772	0.364613	0.650343	0.303583	0.715117		

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



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

best_CCC	best_RSE	model	r	r_se	ci_lwr	ci_upr	v0	v0_rse	r_squared	RSE	CCC	y0	ci_lwr	ci_upr	rho
3	4	Exponential	0.1618389	0.6986109	0.5162882	0.8236294	0.9974091	0.9297077	0.050736	0.357407	0.104580	0.003097	0.35822	0.52198	
2	1	Gompertz	0.0647145	0.4987052	0.0769282	0.8490718	0.6049655	0.822625	0.982493	0.000540	0.000008	0.10358	0.161786		
1	3	Logistic	0.1752045	0.3325742	0.6072078	0.7513663	0.6390218	0.664580	0.452982	0.300081	0.000302	0.10219	0.2024		
4	2	Monomolecular	0.0238960	0.5585310	0.2937562	0.5065673	0.5537531	0.273306	0.359183	0.784868	0.171851	0.889001	1.19479		

epifitter: nonlinear regression

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

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

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

	CCC	r_squared	RSE
Monomolecular	0.8697	0.8020	0.2329
Exponential	0.8587	0.7862	0.2413

	CCC	r_squared	RSE
Gompertz	0.9904	0.9857	0.0683
Logistic	0.9988	0.9981	0.0246

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

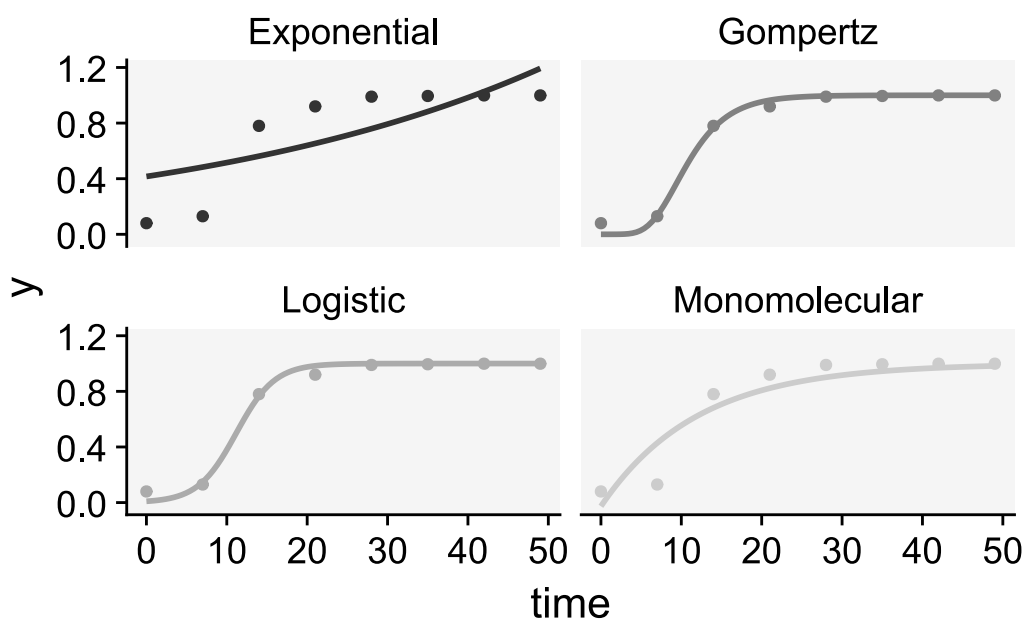
	CCC	r_squared	RSE
Monomolecular	0.8607	0.7699	0.1426
Exponential	0.9880	0.9813	0.0445
Gompertz	0.9946	0.9894	0.0306
Logistic	0.9957	0.9922	0.0270

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

```
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.



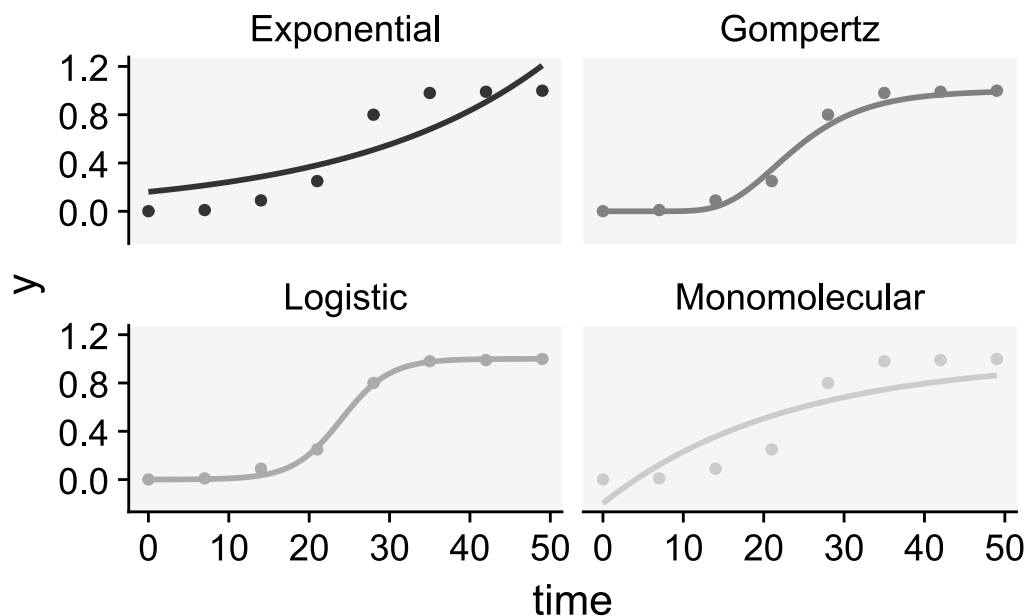
```
knitr::kable(epi11$stats_all)
```


model	y0	y0_se	r	r_sedf	CC	Squared	RSF	ci_lwr	ci_upr	ci_lwr	ci_upr	best_C66	best_RSE
Monomolecular	0.0264211	0.0569336	0.621247	0.933728	0.883222	0.775937	0.481917	0.316303	0.356109	0.418103	0.418103	3	3
Exponential	0.4158016	0.376766	0.150638	0.886587	0.160075	0.034027	0.790087	0.284300	0.001824	0.320000	0.35844	4	4
Gompertz	0.0000000	0.0000000	0.286503	0.631316	0.996209	0.956357	0.388871	0.000000	0.390000	0.209876	0.363131	1	1
Logistic	0.0092495	0.603004	0.181051	0.539931	0.995791	0.993881	0.403179	0.055126	0.240012	0.860085	0.250596	2	2

```
plot_fit(emi22)+
  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.



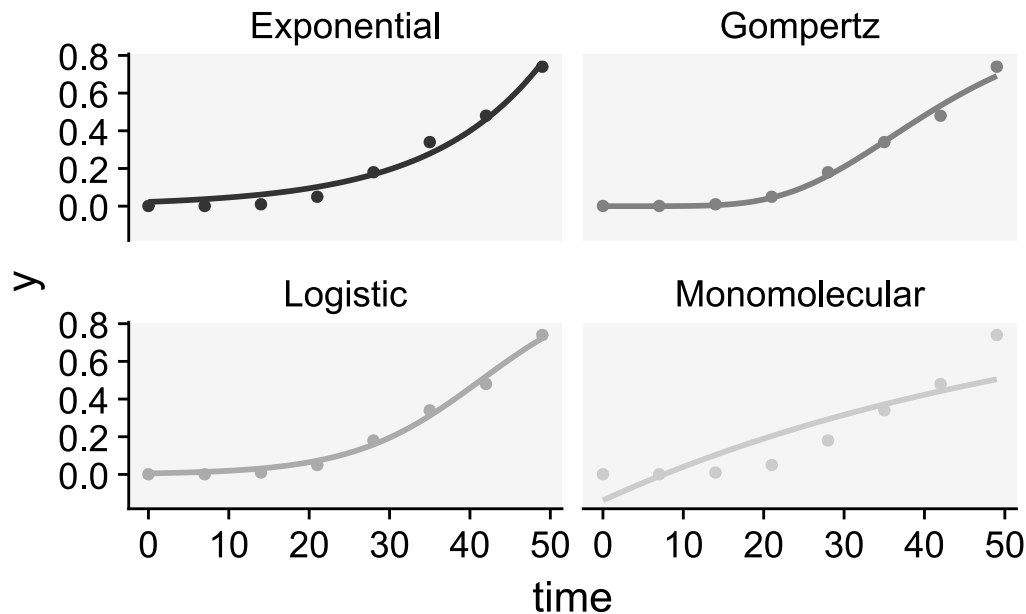
```
knitr::kable(emi22$stats_all)
```

model	y0	y0_se	r	r_sedf	CC	Squared	RSF	ci_lwr	ci_upr	ci_lwr	ci_upr	best_C66	best_RSE
Monomolecular	0.1976541	0.524842	0.613168	0.869673	0.203523	0.288168	0.780925	0.351130	0.119840	0.642802	0.221030	3	3
Exponential	0.4612034	0.483984	0.109366	0.125520	0.858778	0.620424	0.125264	0.463723	0.088103	0.797180	0.768489	4	4
Gompertz	0.0000000	0.0000000	0.161874	0.301525	0.990493	0.985689	0.682506	0.000000	0.000000	0.380927	0.565404	2	2
Logistic	0.0003035	0.020653	0.210402	0.490149	0.998825	0.980692	0.463490	0.001608	0.084082	0.712029	0.306557	1	1

```
plot_fit(emi33)+
  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_sedf	CC	Squared	RSE	FO_ci_lwr	FO_ci_upr	FO_ci_lwr	FO_ci_upr	best_Cost	best_RSE
Monom	0.1874085	0.002921696	0.004253086	0.06957598	0.7692604	0.965922	0.222956	0.656362	0.736690	0.84801	4	4	
Exponential	0.0275267	0.007218471	0.8240704	0.9880069	0.1285445	0.5704863	0.4018905	0.463678	0.891281	119804	3	3	
Gompertz	0.0020002	0.000087598	0.37061025	0.9457498	0.9418705	0.8800007	0.0000002	0.2610615	0.909059	189959	2	2	
Logistic	0.0056634	0.001966224	0.9300867	0.9957129	0.2235927	0.3100852	0.104746	0.377584	0.62102	208322	1	1	

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 specified in the `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_CCC, best_RSE, RSE, CCC)
```

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

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

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

```
# A tibble: 2 × 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)"
)

```

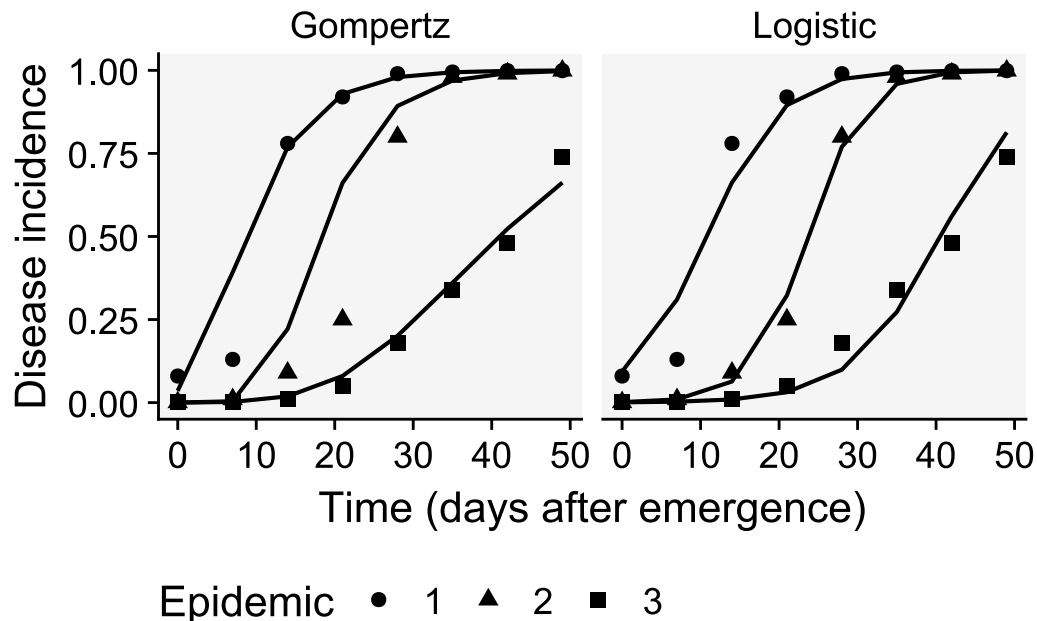


Figure 11.5: Observed (dots) and fitted (line) values for three tobacco etch epidemics in pepper

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"
)

```

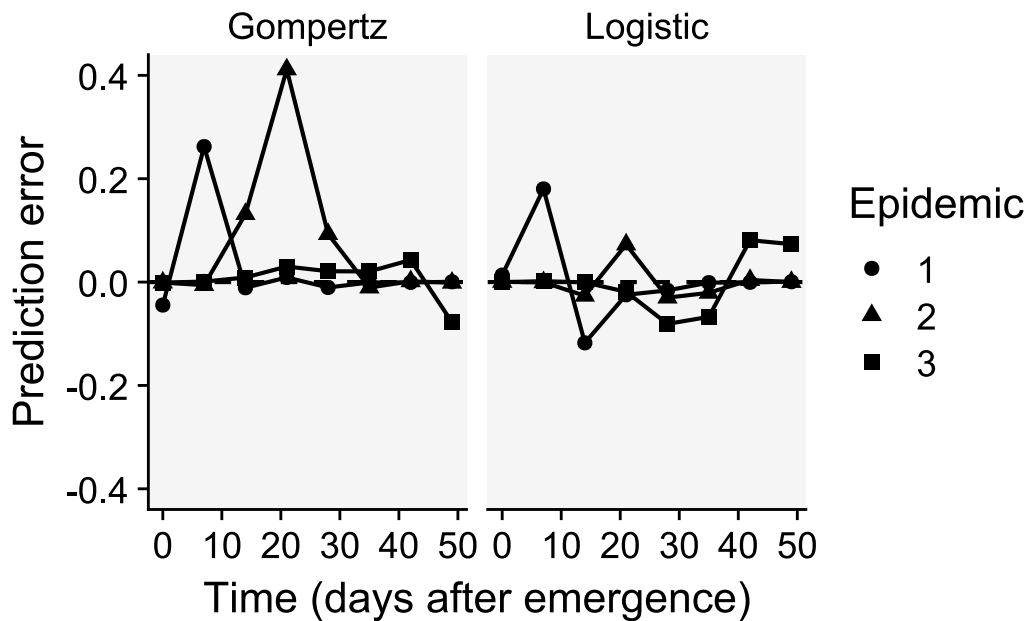


Figure 11.6: Prediction error (dotted lines) by two models fitted to the progress curves of three tobacco etch epidemics in pepper

The plots above confirm 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,
    linewidth = 1
  ) +
  labs(
    x = "Epidemic",
```

```

  y = "r"
)

p2 <- epi_all$Parameters |>
  filter(model == "Logistic") |>
  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,
    linewidth = 1
  ) +
  labs(
    x = "Epidemic",
    y = "y0"
  )
)

library(patchwork)

```

Warning: package 'patchwork' was built under R version 4.5.3

p1 | p2

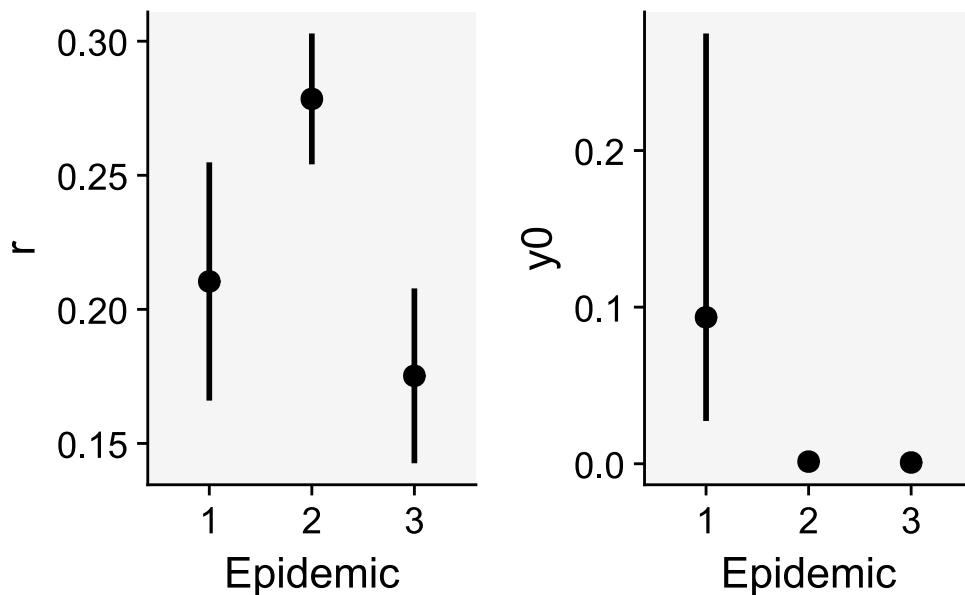


Figure 11.7: Estimated infection rates (left) and initial inoculum (right) by a logistic model fitted to the progress curves of three epidemics of tobacco etch on pepper

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)
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"
```

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

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

```
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 the chosen alpha level (commonly 0.05).

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. These data come 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., 2007b). 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`.

Loading data

```
budblight <- read_csv("https://raw.githubusercontent.com/emdelponte/epidemiology-R/main/data/bud-blight-soybean.csv")
```

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 × 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
```

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
```

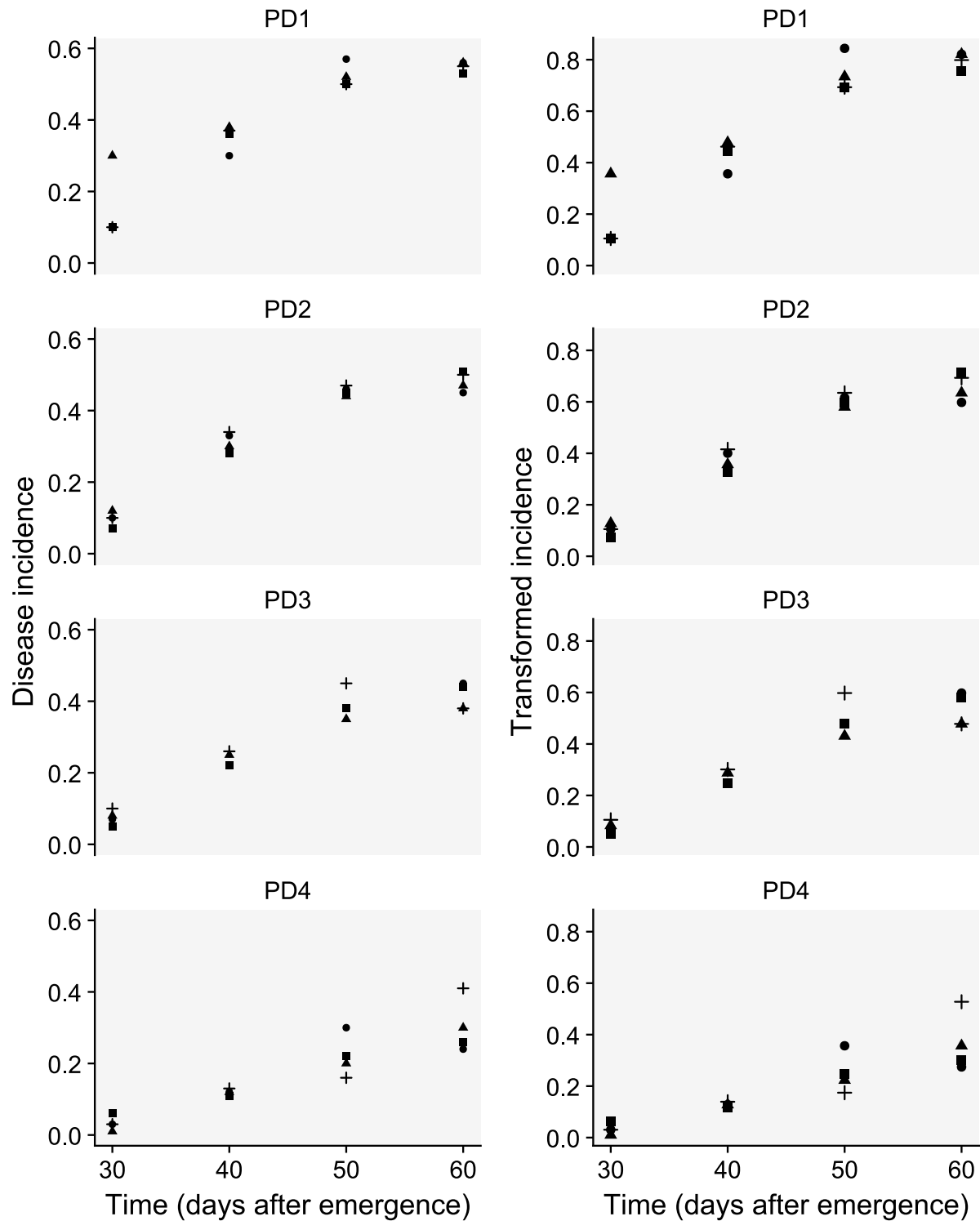


Figure 11.8: Disease progress curves for the incidence of budblight of soybean in Brazil for four planting dates

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. For this exercise, we will use both the linear and the nonlinear estimation method.

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 = buddblight,
  strata_cols = "treat",
  nlin = FALSE
)
```

Let's look at how well the four models fitted the data. Epifitter ranks the four models (1 to 4, where 1 is best) for each treatment, both by CCC (`best_CCC`) and by RSE (`best_RSE`). Let's have a look at the statistics of model fitting.

```
lin1$Parameters |>
select(treat, best_CCC, best_RSE, model, CCC, RSE)
```

	treat	best_CCC	best_RSE	model	CCC	RSE
1	PD1	4	3	Exponential	0.8278055	0.36124036
2	PD1	2	2	Gompertz	0.9040182	0.22226189
3	PD1	3	4	Logistic	0.8711178	0.44751963
4	PD1	1	1	Monomolecular	0.9348429	0.09805661
5	PD2	4	3	Exponential	0.8796705	0.32676216
6	PD2	2	2	Gompertz	0.9307192	0.17938711
7	PD2	3	4	Logistic	0.9062012	0.38773023
8	PD2	1	1	Monomolecular	0.9547434	0.07003116
9	PD3	4	3	Exponential	0.8896173	0.33884790
10	PD3	2	2	Gompertz	0.9288436	0.17156394
11	PD3	3	4	Logistic	0.9085414	0.39051075
12	PD3	1	1	Monomolecular	0.9393356	0.06832499
13	PD4	4	3	Exponential	0.8739618	0.49769642
14	PD4	1	2	Gompertz	0.9234736	0.17474422
15	PD4	3	4	Logistic	0.8911344	0.52412586
16	PD4	2	1	Monomolecular	0.8945962	0.06486949

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)"
  )
)
```

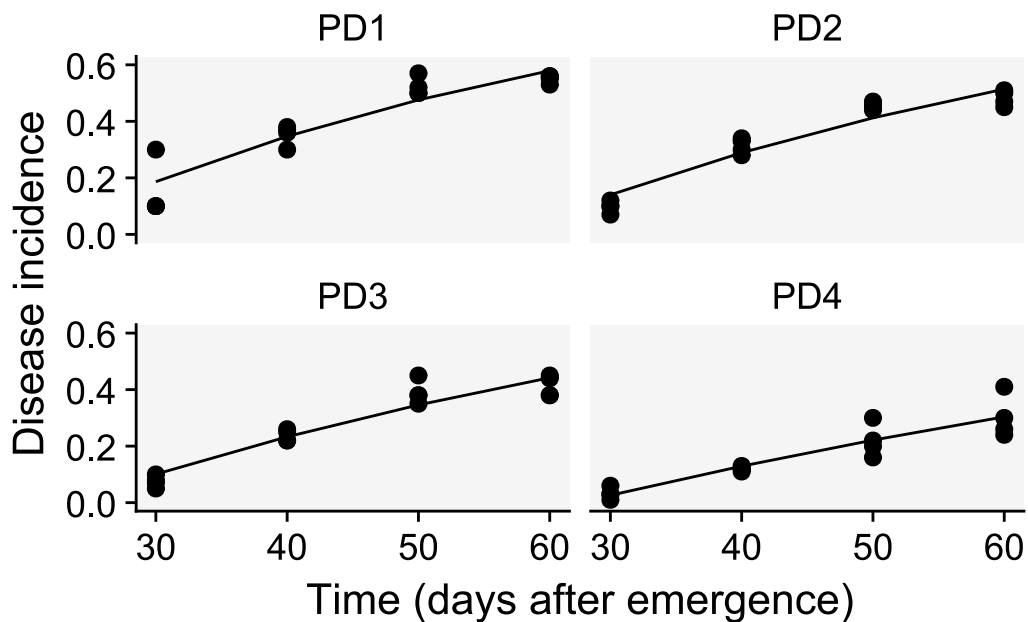


Figure 11.9: Observed (dot) and fitted values by a monomolecular model (line) to the data on the incidence of budblight of soybean in Brazil for four planting dates

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,
    linewidth = 1
  ) +
  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,
    linewidth = 1
  ) +
  labs(
    x = "Time",
    y = "Initial inoculum (y0)"
  )
p5 | p6

```

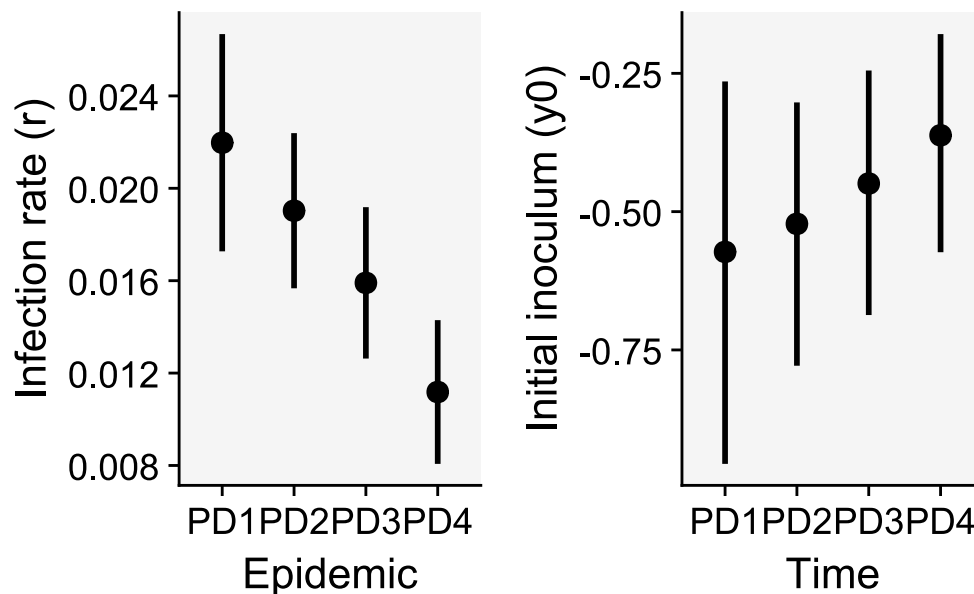


Figure 11.10: Estimates of the infection rate (left) and initial inoculum (right) from the fit of a monomolecular model to the data on the incidence of budblight of soybean in Brazil for four planting dates

Nonlinear 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_CCC, best_RSE)
```

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

And now we obtain the two parameters of interest. Note that the values are not the same 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,
linewidth = 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,
    linewidth = 1
  ) +
  labs(
    x = "Epidemic",
    y = "Initial inoculum (y0)"
  )

```

p7 | p8

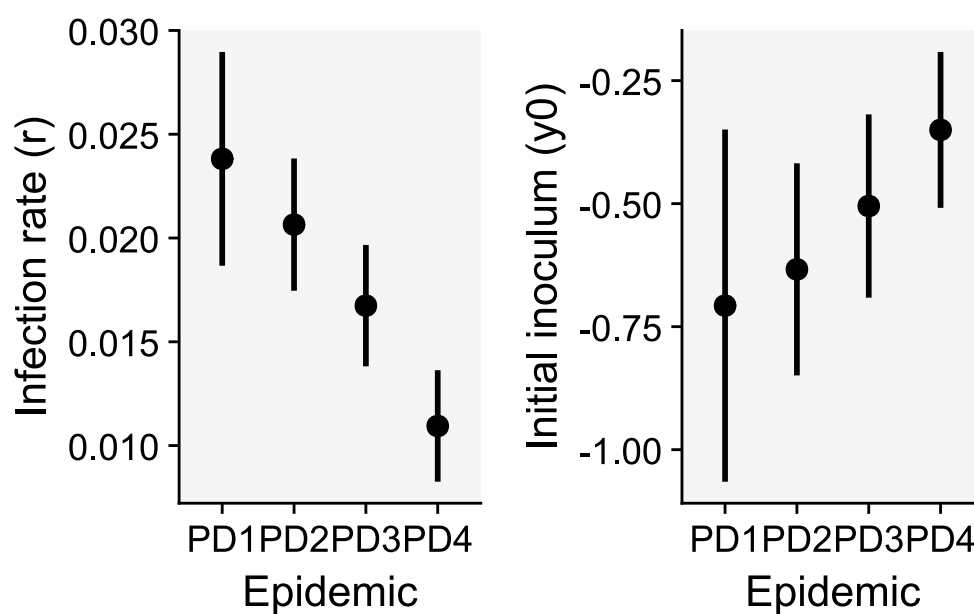


Figure 11.11: Estimates of the infection rate (left) and initial inoculum (right) from the fit of a monomolecular model to the data on the incidence of budblight of soybean in Brazil for four planting dates

12. Compartmental models

Understanding plant disease epidemics requires more than simply describing patterns of spread. Models which represent the biological processes generating those patterns are more informative.

Process-based models do exactly this, by explicitly linking epidemiological dynamics to underlying mechanisms such as transmission and latent infection (Madden et al., 2007a). This provides a framework for interpreting observations, predicting epidemic behaviour and evaluating the likely impact of control strategies.

The dominant paradigm is compartmental modelling, an approach also widely used for diseases of animals and humans (Keeling & Rohani, 2008). In this framework, the host population is divided into a set of mutually exclusive classes ("compartments") based on disease status, with epidemic dynamics represented by the movement of hosts between compartments. For plant diseases, the basic concept goes at least as far back as the seminal work of van der Plank (1963). This simple idea provides a flexible and extensible structure for modelling a wide range of epidemiological systems.

In plant disease, the appropriate choice of unit that is represented by the compartments depends on the biological context (Cunniffe & Gilligan, 2020). For systemic pathogens, such as viruses or some bacteria, it can often be natural to treat individual plants as the fundamental unit. For many diseases, however, and particularly for fungal diseases, infection occurs at much finer scales. Models of these diseases therefore typically instead focus on tissues such as leaves or roots, or even smaller infectable sites (e.g., sections of a leaf which could support a single lesion). At the other extreme, compartmental models can also be used to track disease at much larger spatial scales, for example by taking the quantum to be fields, farms or regions. Such scale-invariance is one of the major strengths of the approach (Madden, 2025).

Beyond their role in representing biological processes, compartmental models are particularly valuable because they form a coherent framework within which the epidemiological impacts of additional complexity can be incorporated and assessed. Biology such as host growth, seasonal cropping cycles, environmental drivers, vectored spread, time-dependent rates, interactions between diseases and disease management interventions can all be introduced by modifying the structure of compartments or the rates of transition between them. Parameters in the model have biological meanings, which allows us to understand and distinguish how different processes affect the epidemiology.

This flexibility is particularly important in plant disease systems, where epidemics often unfold across multiple spatial and temporal scales, and where management decisions must account for both short-term dynamics and longer-term persistence.

As a result, compartmental models are not simply abstract descriptions of epidemics, but practical tools for linking biological understanding to decision-making in real plant disease systems.

In this chapter, we develop this framework progressively. We begin by constructing a simple within-season model, explore its behaviour and key epidemiological properties, and then extend it step by step to incorporate additional biological realism, including primary infection, host growth and multi-season dynamics. Along the way, we show how these models can be implemented and explored using R.

A starting point: the H-L-I-R model

We begin with the simplest useful compartmental model which tracks an epidemic within a single growing season. This serves as a foundation that we will later extend to more complex biological situations.

Biological basis of the model

The key epidemiological difference between individual diseased hosts or units of host tissue is infection status: pre-infectious, infectious or post-infectious. The most obvious compartmental model therefore tracks these three classes of diseased hosts, together with an additional compartment for uninfected healthy hosts.

In plant disease modelling, this formulation is often referred to as the H-L-I-R (Healthy-Latent-Infectious-Removed) model (Madden et al., 2007a). This terminology can obscure the close relationship with the extensive literature on S-E-I-R (Susceptible-Exposed-Infected-Removed) models (Keeling & Rohani, 2008), which are based on precisely the same epidemiological reasoning.

Specifying the model

To construct the model, we first describe the biological processes governing transitions between compartments using word equations, before translating these into mathematical form.

The rates of change are

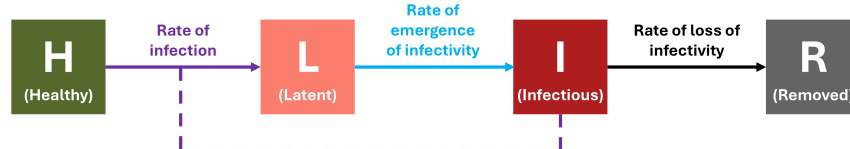
$$\begin{aligned} \text{Rate of change of healthy hosts} &= -\text{Infection}, \\ \text{Rate of change of latently infected hosts} &= \text{Infection} - \text{Emergence of infectivity}, \\ \text{Rate of change of infectious hosts} &= \text{Emergence of infectivity} - \text{Loss of infectivity}, \\ \text{Rate of change of removed hosts} &= \text{Loss of infectivity}. \end{aligned}$$


Figure 12.1: Schematic showing the H-L-I-R compartmental model. The arrows represent the three transitions tracked by the model: infection, emergence of infectivity and loss of infectivity. The dashed line linking the I compartment to the infection transition indicates the rate of this transition depends on the value of I (as well as the value of H).

Transitions come in pairs, since movement between a pair of compartments is being tracked. For example, the Infection transition causes hosts to change status (“move”) from the healthy compartment to the latently infected compartment.

Specifying the model requires us to be more precise about the rates of the three transitions. It is intuitive that, all other things being equal, the net rate at which hosts enter the infectious compartment (the $L \rightarrow I$ transition) should depend only on the number of latently infected hosts. Similarly, the net rate at which infectious hosts are removed (the $I \rightarrow R$ transition) depends only on the number of infectious hosts.

In both cases, rates of these transitions are therefore proportional to the number of hosts which could potentially make the transition, with

$$\begin{aligned} \text{Net rate of emergence of infectivity} &\propto \text{Number of latently infected hosts}, \\ \text{Net rate of loss of infectivity} &\propto \text{Number of infectious hosts}. \end{aligned}$$

The rate at which hosts become latently infected (the $H \rightarrow L$ transition) is more complex. It depends on the current number of healthy hosts, since an individual host can only first become latently infected once. However, since infection is driven by spread from infectious hosts, the rate of the transition should also depend on the number of infectious hosts.

The simplest assumption is to assume bilinearity, i.e., to assume that the rate depends on the product of the number of healthy and infectious hosts. This is sometimes called “mass-action” transmission, and corresponds to homogeneous mixing of hosts, i.e., inoculum from any infectious host is equally likely to cause infection of any other host.

A word equation for the transition rate corresponding to infection is

$$\text{Net rate of infection} \propto \text{Number of healthy hosts} \times \text{Number of infectious hosts}.$$

Finally, note that the interpretation of the removed compartment depends on the biological system. In some pathosystems it may correspond to dead hosts, in others to hosts that remain diseased but are no longer infectious, and in yet others to hosts that have been harvested or otherwise removed from disease transmission. The key feature is that individuals in the removed compartment no longer contribute to new infections.

Writing the model as a system of differential equations

Having specified the transition processes, we can now express the model formally.

Symbol	Meaning
$H(t)$	Healthy hosts at time t
$L(t)$	Latently infected hosts at time t
$I(t)$	Infectious hosts at time t
$R(t)$	Removed hosts at time t

If we introduce the following symbols for parameters

- γ = rate at which latently infected hosts become infectious (per unit time)
- μ = rate at which infectious hosts lose infectivity (per unit time)
- β = rate of infection (per unit time, per infectious host, per healthy host)

then the H-L-I-R model can be expressed as a system of differential equations

$$\begin{aligned}
\frac{dH}{dt} &= -\beta IH, \\
\frac{dL}{dt} &= \beta IH - \gamma L, \\
\frac{dI}{dt} &= \gamma L - \mu I, \\
\frac{dR}{dt} &= \mu I.
\end{aligned}
\tag{12.1}$$

Numbers versus proportions

We have chosen to model the numbers of hosts in each compartment, so that

$$H + L + I + R = N, \tag{12.2}$$

where N is the total host population size.

An alternative formulation is to model the corresponding proportions

$$h = H/N, \quad l = L/N, \quad i = I/N, \quad r = R/N, \tag{12.3}$$

which satisfy

$$h + l + i + r = 1. \tag{12.4}$$

Both approaches contain exactly the same epidemiological information. The choice is largely one of convenience.

Working with numbers can be helpful because it keeps the biological interpretation of the state variables very direct. However, formulations based on proportions are also common, particularly when comparing epidemics in populations of different sizes.

For example, dividing the H-L-I-R equations by N leads to

$$\frac{dh}{dt} = -\beta N i h, \tag{12.5}$$

with analogous equations for the other compartments. In this formulation, the epidemiologically important quantity is often the product βN , rather than β itself.

The important message is that the numbers and proportions formulations contain exactly the same epidemiological information; only the interpretation of the parameters changes.

Meaning of the parameters

Interpreting the parameters biologically connects the mathematical formulation back to observable processes in real systems.

For the per capita rates, γ and μ , there is a well-known inverse relationship between the rate at which an epidemiological event occurs and the time taken on average for the event to occur (Keeling & Rohani, 2008). Here, this means that hosts are latently infected for $1/\gamma$ time units, and the infectious period is $1/\mu$ time units. Consequently, these parameters can be estimated from biological knowledge of the pathogen, for example based upon experiments which involve single hosts in isolation, without tracking interactions or epidemic behaviour.

Conversely, because the infection rate β combines inoculum production, dispersal, host contact and infection efficiency, it is generally not directly measurable in a simple experiment. Moreover, its value depends on factors such as host density, spatial scale and environmental

conditions. Consequently, β is usually estimated by fitting models to epidemic data (see below).

As an aside, note it can sometimes be useful to partition β into its constituent parts, for example as $\beta = \sigma\theta$, where σ relates to the susceptibility of healthy hosts, and θ to the infectivity of infected hosts. For simple models this is largely a matter of presentation, since only the product ever appears in any mathematical expressions. However, making this distinction can be necessary when models are more complex, e.g., when they track pathogen spread on multiple host species with different epidemiological properties (Del Ponte et al., 2026).

Note on host demography

It is helpful to note an important simplifying assumption of the model as currently formulated.

For some pathogens the $I \rightarrow R$ transition corresponds to disease-induced host death, i.e., the pathogen killing a host. It can also often be the case that the host has its own demography, e.g., there is natural production and/or death of hosts.

However, in the model as written above, natural host demography is ignored, i.e., hosts only change status by moving from one epidemiological compartment to the next, with no production of new healthy tissue or (natural) death.

We can see mathematically that the total host population size, $N = H + L + I + R$, is constant in our model by noting

$$\frac{dN}{dt} = \frac{dH}{dt} + \frac{dL}{dt} + \frac{dI}{dt} + \frac{dR}{dt} = 0. \quad (12.6)$$

This is because all transitions between compartments come in pairs, with all outflows from a compartment matched by a corresponding inflow to another compartment.

We show later how within-season growth of hosts can be included in the model.

Symbol	Meaning	Type/Definition
$H(t)$	Number of healthy hosts at time t	State variable
$L(t)$	Number of latent hosts at time t	State variable
$I(t)$	Number of infectious hosts at time t	State variable
$R(t)$	Number of removed hosts at time t	State variable
$P(t)$	Density of primary inoculum at time t	State variable
t	Time	Independent variable
α	Primary infection rate	User-specified parameter
β	Secondary infection rate	User-specified parameter
γ	Rate at which latent hosts become infectious	User-specified parameter
μ	Rate at which infectious hosts become removed	User-specified parameter
ω	Rate of decay of primary inoculum	User-specified parameter
r_H	Host production rate	User-specified parameter
N	Fixed total host population size in the basic H-L-I-R model	$N = H + L + I + R$ (constant)
$T(t)$	Total number of hosts when host growth is included	$T(t) = H(t) + L(t) + I(t) + R(t)$
$D(t)$	Total number of diseased (i.e., non-healthy) hosts	$D(t) = L(t) + I(t) + R(t)$
N_{max}	Carrying capacity	User-specified parameter

Symbol	Meaning	Type/Definition
$T_{\max}$	Length of growing season	User-specified parameter
ϕ	Inoculum production rate	User-specified parameter

Assumptions underpinning the H-L-I-R model

Like all mathematical models, the H-L-I-R model is a simplified representation of reality. Its value lies not in reproducing every biological detail, but in capturing the dominant processes driving epidemic behaviour. Understanding the assumptions of the model is therefore probably just as important as understanding the equations themselves.

The H-L-I-R model makes the following assumptions:

- **Homogeneous mixing.** Every infectious host is assumed to be equally likely to infect every healthy host. The infection rate therefore depends on the product IH . This may be reasonable in well-mixed systems, but becomes less appropriate when spatial structure or local dispersal is important.
- **Identical hosts.** All hosts are assumed to be epidemiologically identical. The model does not distinguish between resistant and susceptible cultivars, young and old hosts, or hosts experiencing different environmental conditions.
- **Fixed parameter values.** The transmission rate (β), latent period ($1/\gamma$) and infectious period ($1/\mu$) are assumed constant throughout the epidemic. In reality these quantities often vary with weather, host development stage and management actions.
- **Deterministic dynamics.** The model predicts the average epidemic trajectory and ignores random events. Consequently, if $R_0 > 1$ (see below), invasion always occurs in the model, whereas chance extinction may occur in real epidemics when infection levels are low.
- **Continuous time.** Hosts move continuously between epidemiological states. This is usually a good approximation when populations are large, although some biological processes may occur at more discrete intervals.
- **Closed population.** No hosts enter or leave the system. As described above, the total host population therefore remains constant, and so the model excludes host growth, harvesting and natural host mortality.

Much of the remainder of this chapter can be viewed as a sequence of examples of progressively relaxing these assumptions. Adding primary inoculum relaxes the assumption of a closed epidemic system. Adding host growth relaxes the assumption of a fixed host population. Extending to multiple seasons relaxes the assumption that epidemics begin and end within a single growing season.

Exploring the model through simulation

Solving the H-L-I-R model numerically

To solve the model numerically, we use the `deSolve` library in R.

To do this, we must specify the numerical values of the parameters; here we take these to be

- $\beta = 0.00003 \text{ host}^{-1} \text{ day}^{-1}$
- $\gamma = 0.2 \text{ day}^{-1}$
- $\mu = 0.1 \text{ day}^{-1}$

Note that since we have cast the model in terms of numbers of hosts, γ and μ have units day⁻¹, whereas β has units per host per day. This is why the product βN , rather than β alone, appears in expressions such as R_0 (see below) or in the equivalent proportion-based formulation.

Given the inverse relationships described above, the choice of γ corresponds to a pathogen with a 5 day latent period, and the choice of μ means the infectious period is 10 days.

Numerical solution also requires initial conditions. We assume there is a single latently infected host in a population of size $N = 10000$, which means

- $H(0) = 9999$
- $L(0) = 1$
- $I(0) = 0$
- $R(0) = 0$

The final component is specifying the range of time over which we are interested in knowing the solution. Here we take $0 \leq t \leq T_{max} = 250$ (days), corresponding to a 250-day period.

The key computational step is the call to the function `deSolve::ode()`, which requires the caller to specify four arguments

- initial condition (`hlir.init`)
- set of times for which a solution is required (`hlir.time`)
- right-hand side of the system of differential equations (`hlir.model`)
- parameters to the system of differential equations (`hlir.params`).

The syntax, perhaps particularly the way parameters are handled by the function `hlir.model`, is somewhat arcane. However, as we shall see, this code provides a template which can, in principle, readily be adapted to any system.

```
library(deSolve)

# specify the model via a function encoding the right hand side
hlir.model <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dH <- - beta * I * H
    dL <- beta * I * H - gamma * L
    dI <- gamma * L - mu * I
    dR <- mu * I
    list(c(dH, dL, dI, dR))
  })
}

# Specify parameters
N <- 10000
T.max <- 250

hlir.params <- c(beta = 0.00003,
                 gamma = 0.2,
                 mu = 0.1)

# Specify initial conditions
hlir.init <- c(H = N - 1,
              L = 1,
              I = 0,
              R = 0)
```

```
# Specify range of times for which a solution is required
hlir.time <- seq(0, T.max, by = 0.01)

# Solve the model
hlir.sol <- ode(y = hlir.init,
               times = hlir.time,
               func = hlir.model,
               parms = hlir.params)

# Cast results to a R data frame
hlir.sol.df <- data.frame(hlir.sol)

# Plot results using our custom helper function
plot.model.results(hlir.sol.df, type="HLIR")
```

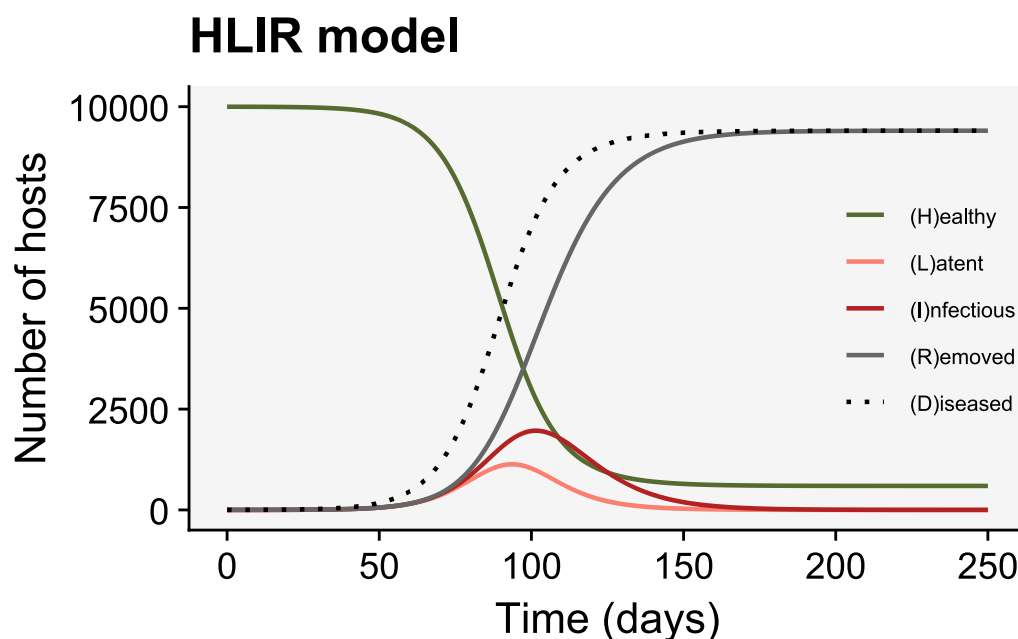


Figure 12.2: Numerical solution of the H-L-I-R model: numbers of healthy (H), latently infected (L), infectious (I) and removed (R) hosts, and the number of diseased hosts (D), through time

Interpreting epidemic behaviour

The simulation shows the characteristic behaviour of an epidemic. Initially, disease prevalence increases slowly, but then the epidemic accelerates, causing a rapid decline in healthy hosts. Eventually, depletion of healthy hosts limits transmission and the epidemic dies out, with most (but not all) hosts entering the removed class.

The key features to notice are the lag before infectious tissue builds up, the subsequent rapid loss of healthy hosts, and the eventual plateau in total diseased tissue.

The number of diseased hosts, $D(t)$, is also shown, where

While numerical simulation provides a detailed picture of epidemic behaviour, it is often useful to summarise these dynamics using a small number of quantitative measures.

In particular, we may wish to answer questions such as:

- Under what conditions can an epidemic invade?

- How large will the epidemic become?
- How quickly will infection spread initially?

Compartmental models allow these questions to be answered analytically through a set of standard epidemiological quantities and calculations, which we now introduce.

Key epidemiological quantities

Certain calculations are routine using compartmental models, and provide concise summaries of key features of epidemic dynamics. We illustrate these for the H-L-I-R model here.

Basic reproduction number

The first question is whether a pathogen can successfully invade a healthy host population.

In the H-L-I-R model an individual host's infectious period is finite and therefore a pathogen cannot invade if its net rate of infection is too low relative to how long it stays infectious.

This important idea is captured in a key threshold, the basic reproduction number, R_0 , the number of new infections resulting from the introduction of a single infected host into an otherwise healthy population.

There are a number of methods of different levels of mathematical sophistication for finding R_0 . For simple models, it can be found heuristically.

Assuming the total size of the population is

$$H + L + I + R = N, \quad (12.7)$$

then at the start of a potential epidemic initiated by a single infected host,

$$L = R = 0, \quad (12.8)$$

and

$$I = 1, \quad (12.9)$$

and

$$H = N - 1 \approx N. \quad (12.10)$$

Considering only the infections directly caused by the single infected host and multiplying the net rate at which healthy hosts become infected by the host's average infectious period implies This is because the infected host generates infections at total rate βN and remains infectious for $1/\mu$ time units.

If $R_0 > 1$ then the pathogen will invade and the amount of infection will increase over time, at least initially.

Note that the parameter controlling the rate of emergence of infectivity (γ) is not involved in the expression for R_0 , i.e., the pathogen's basic reproduction number is independent of its latent period. This makes sense, because in this model the latent period only affects how rapidly hosts are infected, rather than how many infections are caused. Note this would no longer be true if natural death were included in the model, since then we would need to account for the probability of a latently infected host successfully becoming infectious rather than dying naturally. In the current model, although hosts spend time in the latent compartment before becoming infectious, every latently infected host is guaranteed to

eventually reach the infectious compartment. The latent period therefore alters the timing of transmission but not the number of secondary infections generated.

For this parameterisation of the H-L-I-R model, $R_0 = 3$.

```
hlir.r0.func <- function(beta,N,mu)
{
  return(beta * N / mu)
}

hlir.r0 <- hlir.r0.func(hlir.params[["beta"]],
                        N,
                        hlir.params[["mu"]])

print(paste("R0 =",hlir.r0))
```

```
[1] "R0 = 3"
```

Final size equation

While R_0 determines whether an epidemic can invade, it does not tell us how large the epidemic will become once it is established.

Not all hosts necessarily become infected by the end of the epidemic. This can be quantified by the final size equation (Keeling & Rohani, 2008), which states that the fraction of the population which eventually becomes infected, i.e.,

$$f = 1 - H(\infty)/N \quad (12.11)$$

satisfies

$$f = 1 - \exp(-fR_0). \quad (12.12)$$

Note that because in this model at the end of the epidemic $L(\infty) = I(\infty) = 0$, all infected hosts are ultimately found in the removed compartment, and so

$$f = R(\infty)/N. \quad (12.13)$$

The final size equation is a transcendental equation, and so must be solved numerically.

The code below uses the base R function `uniroot` to solve it in R, and compares this with the final value of R as obtained from the numerical solution at $t = T_{max} = 250$ days.

```
hlir.final.size <- function(r0)
{
  f <- function(z) z - (1 - exp(-r0 * z))

  ret.val <- 0
  if(r0 > 1)
  {
    ret.val <- uniroot(f, lower = 1e-10, upper = 1)$root
  }
  return(ret.val)
}

hlir.f <- hlir.final.size(hlir.r0)
print(paste("From final size equation: f =", hlir.f))
```

```
[1] "From final size equation: f = 0.940479945459519"
```

```
print(paste("From numerical solution: f =", h1ir.sol.df[which.max(h1ir.sol.df$time), "R"]/N))
```

```
[1] "From numerical solution: f = 0.940466373627097"
```

In both cases, approximately 94% of all hosts are infected by the end of the epidemic, with a very small discrepancy between the two calculations because the epidemic has not quite completed within $T_{max} = 250$ days.

Initial rate of epidemic increase

A complementary question concerns the speed of epidemic growth during the early stages of spread.

When disease is rare, almost all hosts are healthy, so the epidemic is operating near the disease-free equilibrium. In this regime, the nonlinear model behaves approximately like a linear system. The dominant behaviour of that linear system determines whether infection grows or declines, and how rapidly this occurs. The quantity r is the corresponding exponential growth rate.

This initial exponential rate of disease increase can be found (Cunniffe et al., 2012) by calculating the largest real part of the eigenvalues of the Jacobian matrix of the system evaluated at the disease-free-equilibrium

$$(H, L, I, R) = (N, 0, 0, 0) \quad (12.14)$$

The Jacobian matrix is simply a mathematical object that captures how small perturbations to each compartment affect the rates of change of all other compartments.

This can be done using the function `jacobian.full()` in the `rootSolve` package.

```
library(rootSolve)

# disease free equilibrium
h1ir.dfe <- c(H = N,
             L = 0,
             I = 0,
             R = 0)

# find jacobian matrix at h1ir.dfe
h1ir.jac <- jacobian.full(y = h1ir.dfe,
                        func = h1ir.model,
                        parms = h1ir.params)

# find eigenvalues and eigenvectors
h1ir.eigen.jac <- eigen(h1ir.jac)

# extract and print the largest eigenvalue
print(paste("r =", max(h1ir.eigen.jac[["values"]])))
```

```
[1] "r = 0.1"
```

Note that the calculation done by the code above involving the function `rootSolve::jacobian.full()` would in fact be the same for any compartmental model, and so does not require any

mathematics. You do not need to understand the linear algebra here. The software does the calculation automatically.

That the numerical solution matches these theoretical expectations can be checked.

On the figures below, the left hand panel shows the logarithm of the number of infectious hosts as a function of time. There is an initial transient (for very small values of t) followed by an extended period (from around 10 to 100 days) within which the logarithm of the number of infectious hosts grows linearly, and so in which the number of infectious hosts is growing exponentially.

(Arbitrarily) selecting $t_{start} = 20 \leq t \leq 50 = t_{end}$ as a representative region within this range of plausibly exponential increase, a subset of the curve is plotted on the right-hand side (black), together with the results of a linear regression (red).

```
library(ggplot2)
library(r4pde)
library(patchwork)
```

Warning: package 'patchwork' was built under R version 4.5.3

```
# First plot: log(I) vs t
p1 <- ggplot(hlir.sol.df, aes(x = time, y = log(I))) +
  geom_line(linewidth = 0.8) +
  labs(
    x = "time (t)",
    y = "log(Infectious)",
    subtitle = "log(I) vs t"
  ) +
  theme_r4pde()

# extract just t and I for 20 <= t <= 50
t.start <- 20
t.end <- 50
t.subset <- hlir.sol.df$time[hlir.sol.df$time>=t.start&hlir.sol.df$time<=t.end]
i.subset <- hlir.sol.df$I[hlir.sol.df$time>=t.start&hlir.sol.df$time<=t.end]

subset_df <- data.frame(
  time = t.subset,
  log_I = log(i.subset)
)

# fit a linear model and report its gradient
my.mod <- lm(log(i.subset)~t.subset)
print(paste("Fitted r =", coef(my.mod)[[2]]))
```

```
[1] "Fitted r = 0.0995145118548592"
```

```
# Predict values for the regression line
subset_df$pred <- predict(my.mod)

# Second plot: subset with regression line
p2 <- ggplot(subset_df, aes(x = time)) +
  geom_line(aes(y = log_I, color = "Simulated"), linewidth = 0.8) +
  geom_line(aes(y = pred, color = "Fitted"), linetype = "dashed", linewidth = 0.8) +
```

```

scale_color_manual(
  name = NULL,
  values = c("Simulated" = "black", "Fitted" = "red")
) +
labs(
  x = "time (t)",
  y = "log(Infectious)",
  subtitle = paste("log(I) vs t for", t.start, "<= t <=", t.end)
) +
theme_r4pde() +
theme(legend.position = c(0.25, 0.8))

p1 + p2

```

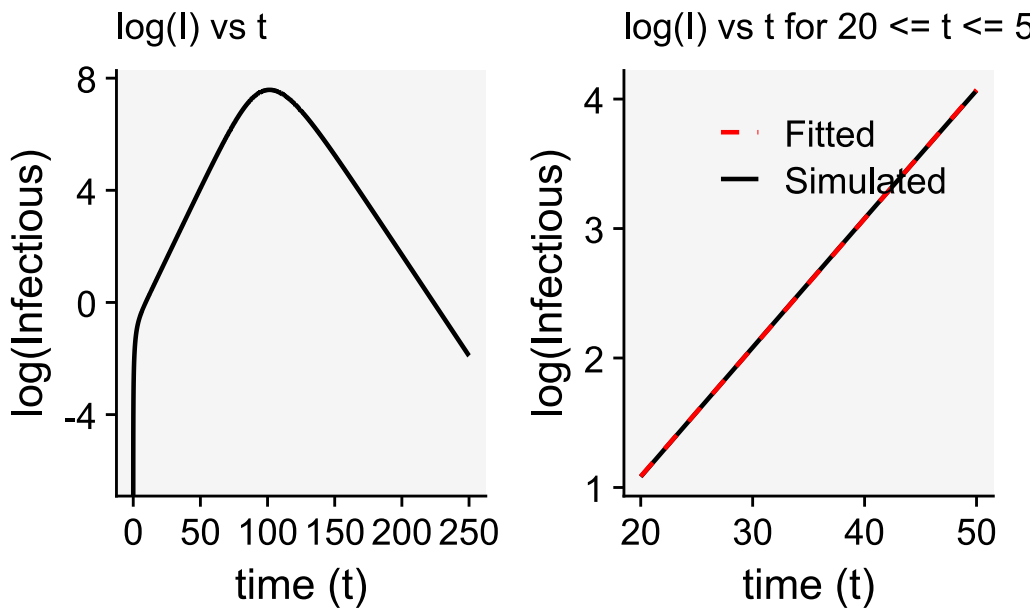


Figure 12.3: Left: logarithm of the number of infectious hosts through time. Right: the segment used to estimate the initial exponential growth rate (black) and the fitted linear regression (red, dashed)

Note the initial exponential growth rate matches to two significant figures.

Together, these quantities — R_0 , the final size, and the initial growth rate — provide complementary summaries of epidemic behaviour, linking the structure of the model to biologically meaningful outcomes.

Although both R_0 and r describe epidemic invasion, they capture different aspects of epidemic behaviour. R_0 measures how many secondary infections are generated, whereas r measures how rapidly infection increases through time.

Extending the H-L-I-R framework: adding biological realism

Adding primary infection (H-L-I-R-(P) model)

To illustrate the flexibility of the compartmental framework, as well as to introduce further biology which will be needed to model multi-seasonal epidemics, primary infection is added to

the model. This corresponds to infections caused by inoculum produced elsewhere (i.e., infections which are not direct secondary infections from infected hosts within the system of interest)

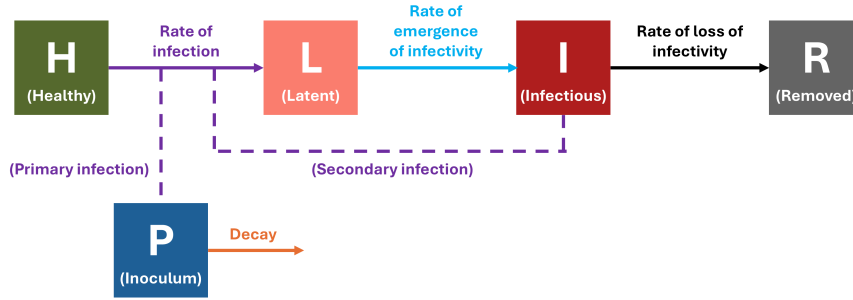


Figure 12.4: Schematic showing the H-L-I-R-(P) compartmental model. A new compartment has been added, which tracks the density of inoculum (i.e., this compartment does not relate to host status), which decays over time, but which infects healthy hosts. The two types of infection – primary and secondary – are distinguished on the diagram.

One way of doing this involves introducing an additional compartment, P , which tracks the density of primary inoculum. We assume that primary inoculum only decays, but also leads to primary infection (and so a transition of hosts from $H \rightarrow L$). Primary infection is assumed to occur at a rate which is jointly proportional to the density of inoculum and the number of healthy hosts, i.e.,

Note that there are now two infection pathways, the net result of which sets the overall rate of infection

- Primary infection. Infection from outside the system (i.e., from the inoculum)
- Secondary infection. Infection from within the system (i.e., from infectious hosts)

If we assume the inoculum (P) decays at characteristic rate ω (per day per unit density of inoculum) and leads to primary infection at rate α (per unit density of inoculum per healthy host), the equations of the H-L-I-R-(P) model are

$$\begin{aligned}
 \frac{dH}{dt} &= -\alpha PH - \beta IH, \\
 \frac{dL}{dt} &= \alpha PH + \beta IH - \gamma L, \\
 \frac{dI}{dt} &= \gamma L - \mu I, \\
 \frac{dR}{dt} &= \mu I, \\
 \frac{dP}{dt} &= -\omega P.
 \end{aligned} \tag{12.15}$$

This model can be simulated too.

Relative to our previous template, we need to

- define a new function for the right hand side of the model
- update the parameters to include ω and α
- update the initial condition to include $P(0)$

Of course, numeric values for the two new parameters and one new initial condition are required. In the code below we set

- $\omega = 0.05 \text{ day}^{-1}$
- $\alpha = 0.00001 \text{ inoculum}^{-1} \text{ day}^{-1}$
- $P(0) = 10$

We also set

- $H(0) = 10000$
- $L(0) = 0$
- $I(0) = 0$
- $R(0) = 0$

which corresponds to assuming no hosts are infected at $t = 0$.

We have therefore assumed the inoculum has an average lifetime of $1/\omega = 20$ days, and that there are 10 units of inoculum present at $t = 0$.

The value of α requires some interpretation. It is the infection rate of healthy hosts per unit inoculum per host per unit time. So, in a system with $N = 10000$ hosts, with $P(0) = 10$, we would expect the initial net rate of primary infection to be $N\alpha P(0) = 1$, and so for an average of one host to become infected via the primary infection pathway per day.

However, because the decay rate of inoculum is relatively large, this drops off quite rapidly (and is further reduced because as time goes by there are fewer healthy hosts to infect).

The code to simulate the model is given below. Note it is just a lightly edited version of the code we originally used to simulate the H-L-I-R model.

```
hlirp.model <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dH <- - alpha * P * H - beta * I * H
    dL <- alpha * P * H + beta * I * H - gamma * L
    dI <- gamma * L - mu * I
    dR <- mu * I
    dP <- - omega * P
    list(c(dH, dL, dI, dR, dP))
  })
}

hlirp.params <- hlir.params
hlirp.params[["omega"]] <- 0.05
hlirp.params[["alpha"]] <- 0.00001

hlirp.init <- c(H = N,
               L = 0,
               I = 0,
               R = 0,
               P = 10)

hlirp.time <- hlir.time

# Solve the model
hlirp.sol <- ode(y = hlirp.init,
               times = hlirp.time,
               func = hlirp.model,
               parms = hlirp.params)
```

```
# Cast results to a R data frame
hlirp.sol.df <- data.frame(hlirp.sol)

# Plot results using our custom helper function
plot.model.results(hlirp.sol.df, type="HLIRP")
```

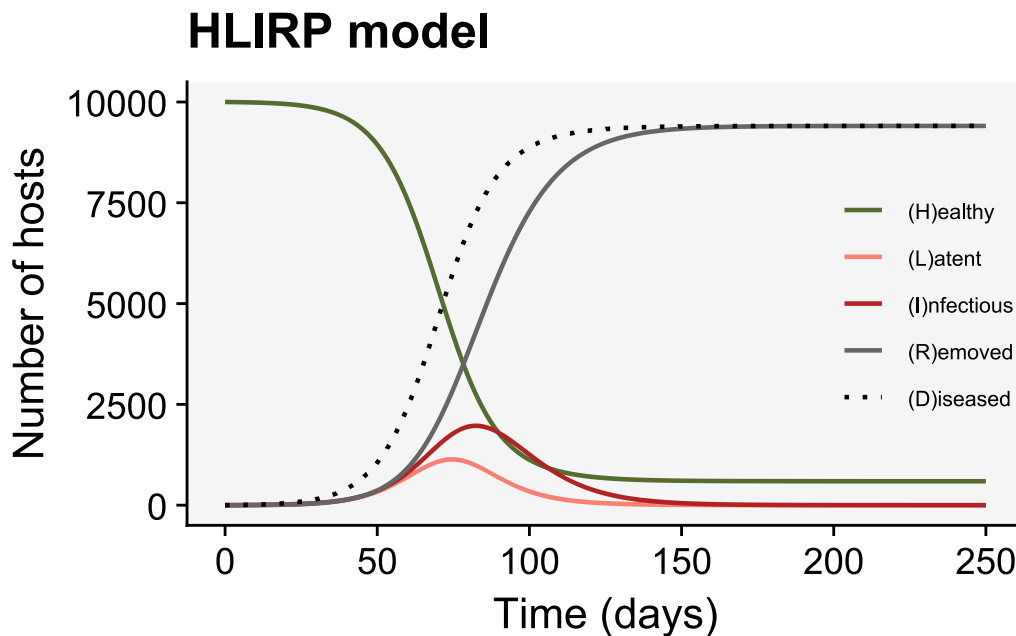


Figure 12.5: Numerical solution of the H-L-I-R-(P) model, which adds primary infection from a decaying pool of inoculum

Compared with the previous simulation, primary inoculum removes the need for initial infected hosts, allowing the epidemic to start even when $L(0) = I(0) = R(0) = 0$.

Note that for these parameters, the epidemic gets going slightly more rapidly.

This formulation is useful not only for representing infection from outside the focal population, but also because it provides a mechanism by which inoculum carried over from one season can seed infection in the next (see below).

Adding host growth

Allowing host demography changes the balance between infection and replenishment, potentially altering both the growth rate and long-term behaviour of the epidemic.

We switch focus here, and focus on a crop plant host, which produces more infectable tissue over the course of the growing season (e.g., a wheat plant producing more leaves over time). It then makes sense to imagine the amount of host tissue being subject to its own dynamics, with an increase in the amount of tissue over time.

In what follows we will continue to refer to “hosts” for simplicity, even though now we are actually thinking about the number/density of infectable sites rather than whole plants.

There are various ways of modelling host growth. We introduce the idea in a simple way here, with logistic growth of healthy hosts at rate r_H , to a carrying capacity N_{max} . Adding this to the H-L-I-R-(P) model leads to

$$\begin{aligned}
\frac{dH}{dt} &= r_H H \left(1 - \frac{H + L + I + R}{N_{max}} \right) - \alpha P H - \beta I H, \\
\frac{dL}{dt} &= \alpha P H + \beta I H - \gamma L, \\
\frac{dI}{dt} &= \gamma L - \mu I, \\
\frac{dR}{dt} &= \mu I, \\
\frac{dP}{dt} &= -\omega P.
\end{aligned} \tag{12.16}$$

We have included the total amount of host tissue in the density-dependent term in the logistic growth, which corresponds to assuming all classes of host take up space and/or consume resources, whereas we have only included the number of units of healthy host in the production term. Depending on the system being modelled, it could, perhaps, be argued that R should be omitted from the density dependence, or L be included in the production term. It might also be appropriate to include natural exponential death terms on all state variables, to account for natural death of host tissue, but we have not done so here.

The updated model can again be solved by adapting our previous template. The following new parameter values are required

- $r_H = 0.05 \text{ day}^{-1}$
- $N_{max} = 10000$

In the original H-L-I-R model, the total host population size was fixed, and we denoted this constant by

$$N = H + L + I + R \tag{12.17}$$

Once host growth is added, however, this quantity is no longer constant. To avoid confusion, we denote the time-varying total number of hosts by

$$T(t) = H(t) + L(t) + I(t) + R(t). \tag{12.18}$$

The carrying capacity for this total amount of host tissue is denoted by N_{max} . In the figure below, $T(t)$ is shown by the dotted blue line. Also note that we start with $H(0) = N_{max}/20$, i.e., a small number of hosts initially.

```

hlirp.with.growth.model <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {
    dH <- r.H * H * (1 - ((H + L + I + R)/N.max)) - alpha * P * H - beta * I * H
    dL <- alpha * P * H + beta * I * H - gamma * L
    dI <- gamma * L - mu * I
    dR <- mu * I
    dP <- - omega * P
    list(c(dH, dL, dI, dR, dP))
  })
}

hlirp.with.growth.params <- hlirp.params
hlirp.with.growth.params[["N.max"]] <- 10000
hlirp.with.growth.params[["r.H"]] <- 0.05

hlirp.with.growth.init <- c(H = hlirp.with.growth.params[["N.max"]]/20,
  L = 0,

```



```

I = 0,
R = 0,
P = 10)

hlirp.with.growth.time <- hlir.time

# Solve the model
hlirp.with.growth.sol <- ode(y = hlirp.with.growth.init,
                             times = hlirp.with.growth.time,
                             func = hlirp.with.growth.model,
                             parms = hlirp.with.growth.params)

# Cast results to a R data frame
hlirp.with.growth.sol.df <- data.frame(hlirp.with.growth.sol)

# Plot results using our custom helper function
plot.model.results(hlirp.with.growth.sol.df, type="HLIRP (with host growth)")

```

HLIRP (with host growth) model

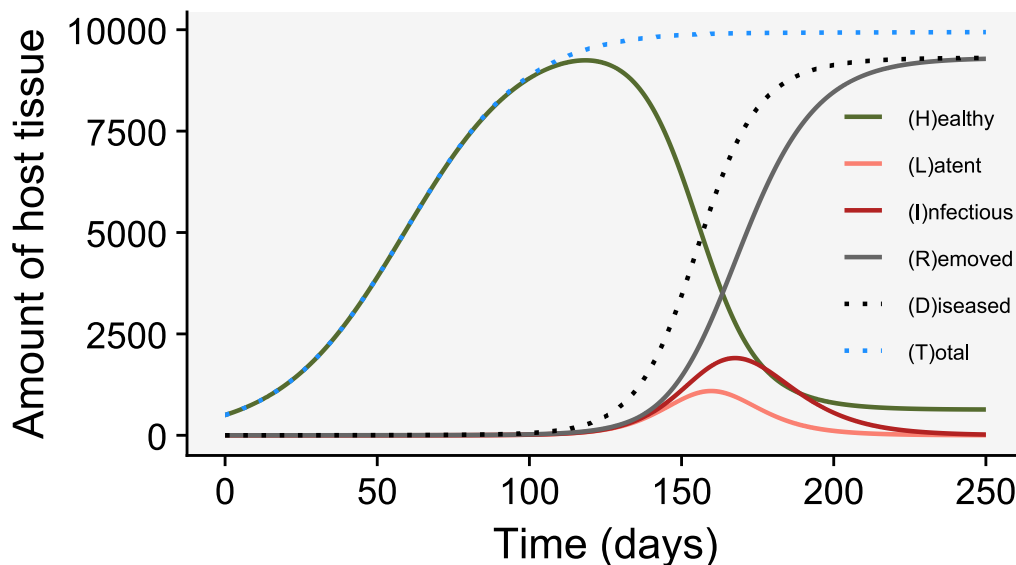


Figure 12.6: Numerical solution of the H-L-I-R(P) model with host growth. The dotted blue line is the total amount of host tissue

Note the epidemic is much slower and has only just completed by $t = T_{max}$. This is because the epidemic only becomes self-sustaining when the amount of healthy host tissue is large enough that the effective reproduction number exceeds one. In this model, when infection is rare, a natural approximation to the instantaneous reproduction number is $\beta H(t)/\mu$. Early in the season $H(t)$ is low, so infection fails to increase even though host growth later creates enough healthy tissue for spread.

Accounting for multiple seasons

Extending the model across seasons introduces new mechanisms for persistence that are not captured by within-season quantities such as R_0 alone.

Polyetic epidemics

A key distinguishing feature of crop disease epidemiology is the regular perturbations introduced by harvesting and replanting (Cunniffe et al., 2015). Many crop disease epidemics are polyetic, unfolding over multiple growing seasons (Zadoks, 1974). Obvious examples are diseases caused by soil-borne pathogens, or by other pathogens with long-lived survival structures. In yet other systems, an epidemic within the crop leads to infection of nearby volunteer plants or other vegetation, which serves as a natural reservoir of inoculum. In all of these systems, infection does not simply run its course and disappear within a single year; instead, inoculum produced in one season seeds epidemics in subsequent seasons.

We update our model to represent a polyetic crop epidemic over multiple growing seasons, again adding complexity gradually. This final model is loosely based on the models given in Sections 6.2.2 and 6.3 of the Madden et al. (2007a) textbook, which themselves were based on the model presented in Madden & van den Bosch (2002).

However, we note there are many examples of models of polyetic epidemics in the plant disease modelling literature, ranging from models targeting biological control (Gubbins & Gilligan, 1997), to fungicide resistance (Elderfield et al., 2018), to pathogen speciation (Hamelin et al., 2011), to integrated pest management (Vincent et al., 2025), to host plant resistance (Fabre et al., 2012), to fluctuations over years caused by stochastic emergence and re-emergence (Willoquet et al., 2020). All of these models of polyetic epidemics have the same fundamental ideas at their core; tracking the pathogen over successive growing seasons and modelling the process of inoculum carry-over from one season to the next.

Linking seasons via inoculum carryover

The H-L-I-R-(P) model as presented above implicitly describes a single-season epidemic. To extend this framework to multiple seasons, we must represent how infection or inoculum persists between seasons.

A simple way to represent multiple seasons is to assume that each season begins with a fresh population of healthy hosts, but with some carryover of infection pressure from the previous season via the inoculum compartment (P).

Let $I_k(t)$ denote the time-dependent amount of infectious hosts during season k . Within each season, the epidemic follows an H-L-I-R-(P) model, but the initial amount of inoculum depends on what happened in the previous season.

For example, we might assume:

- a constant fraction of hosts are replanted at the start of each season, so $H_k(0) = N_{max}/20$,
- no hosts are actively infected at planting, so $L_k(0) = I_k(0) = R_k(0) = 0$,
- the initial amount of primary inoculum in season k , $P_k(0)$, depends on inoculum surviving from season $k - 1$.

A simple model of inoculum production and carryover between seasons is

$$P_k(0) = \phi \int_0^{T_{max}} I_{k-1}(t) dt, \quad (12.19)$$

where:

- T_{max} is the length of the growing season,
- ϕ is rate at which infectious material is turned into inoculum (that survives between seasons).

Similarly to how we have treated host growth, more complex models of this component are possible, and here we have chosen to take a simple model that illustrates the underpinning idea.

The seasonal epidemic is represented by a continuous-time compartmental model, while the carry-over of inoculum is represented by a discrete-time map between seasons.

Within each season, the dynamics are as before, but the model must track the amount of inoculum produced. The following code introduces a function which does this using the results from the H-L-I-R-(P) model run above, doing the numerical integration via the function

`sfsmisc::integrate.xy`.

```
library(sfsmisc)
```

```
Attaching package: 'sfsmisc'
```

```
The following object is masked from 'package:dplyr':
```

```
last
```

```
find.inoc <- function(data, phi)
{
  inoc <- phi * integrate.xy(data$time, data$I)
  return(inoc)
}
phi <- 0.00001
print(find.inoc(hlirp.with.growth.sol.df, phi))
```

```
[1] 0.928291
```

Simulating multiple seasons

Having specified the mapping between successive seasons, the model is sufficiently “rich” to be able to track a multi-seasonal epidemic. We simulate over 5 years, starting from a very small initial amount of inoculum in season one, with $P_1(0) = 0.0001$.

The top plot shows the amount of disease, $D(t)$ and the total amount of host tissue, $T(t)$ over successive seasons. The responses of the individual compartments (e.g., $H(t)$, $L(t)$, $I(t)$ and $R(t)$) are not shown to avoid cluttering the figure.

The bottom plot shows how the initial amount of inoculum, $P_k(0)$ changes over time. This is shown on a logarithmic scale to allow small changes to be seen.

```
n.yr <- 5
init.inoc <- 0.0001
phi <- 1e-4
multi.to.plot <- list(n.yr = n.yr,
                      n.max = hlirp.with.growth.params[["N.max"]],
                      inoc = numeric(n.yr),
                      results = list())

for(i in 1:n.yr)
{
  # Update initial conditions
  hlirp.with.growth.init <- c(H = hlirp.with.growth.params[["N.max"]]/20,
```

```

      L = 0,
      I = 0,
      R = 0,
      P = init.inoc)

# Store in the results structure
multi.to.plot$inoc[i] <- init.inoc

# Solve the differential equations
hlirp.with.growth.sol <- ode(y = hlirp.with.growth.init,
                             times = hlirp.with.growth.time,
                             func = hlirp.with.growth.model,
                             parms = hlirp.with.growth.params)

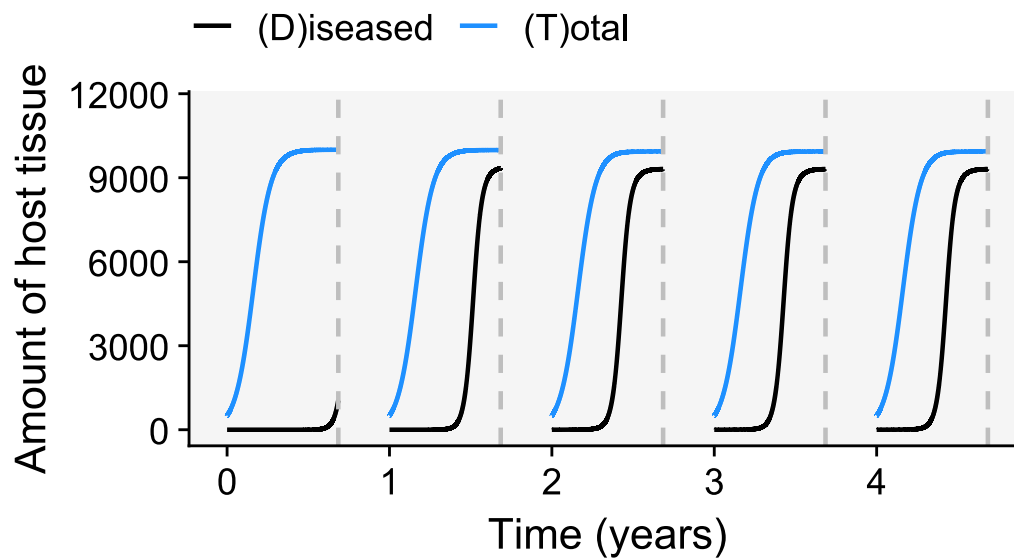
# Cast results to a R data frame
hlirp.with.growth.sol.df <- data.frame(hlirp.with.growth.sol)

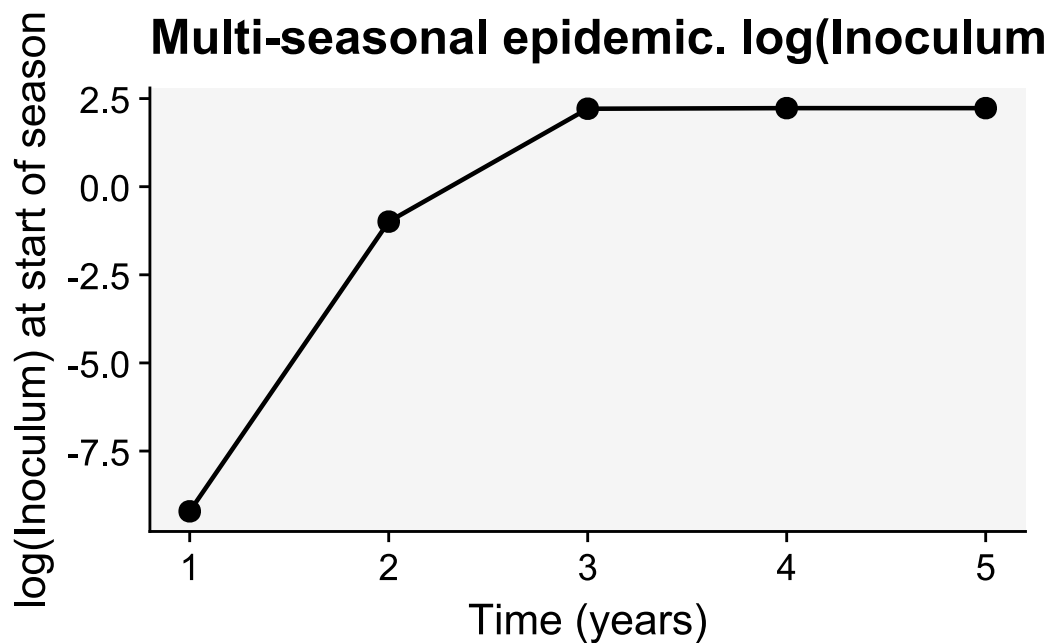
# Store in the results structure
multi.to.plot$results[[i]] <- hlirp.with.growth.sol.df

# Calculate how much inoculum was produced this season
init.inoc <- find.inoc(hlirp.with.growth.sol.df, phi)
}
plot.model.results(multi.to.plot,"MULTI")

```

Multi-seasonal epidemic. Disease prc





In the first year the small amount of inoculum leads to only a very small epidemic, although this does lead to a larger amount of inoculum at the start of year two. This then leads to a bigger epidemic, which produces an even larger amount of inoculum. After three years the system settles onto a repeated cycle, with the same amount of inoculum at the beginning of each year, and a correspondingly similar (and large) epidemic each year.

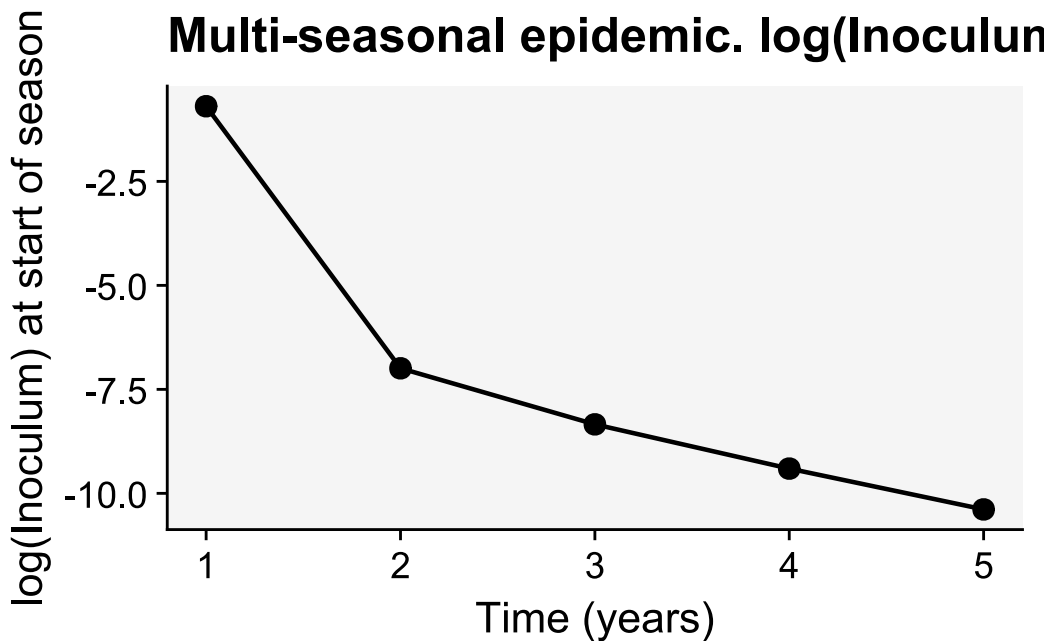
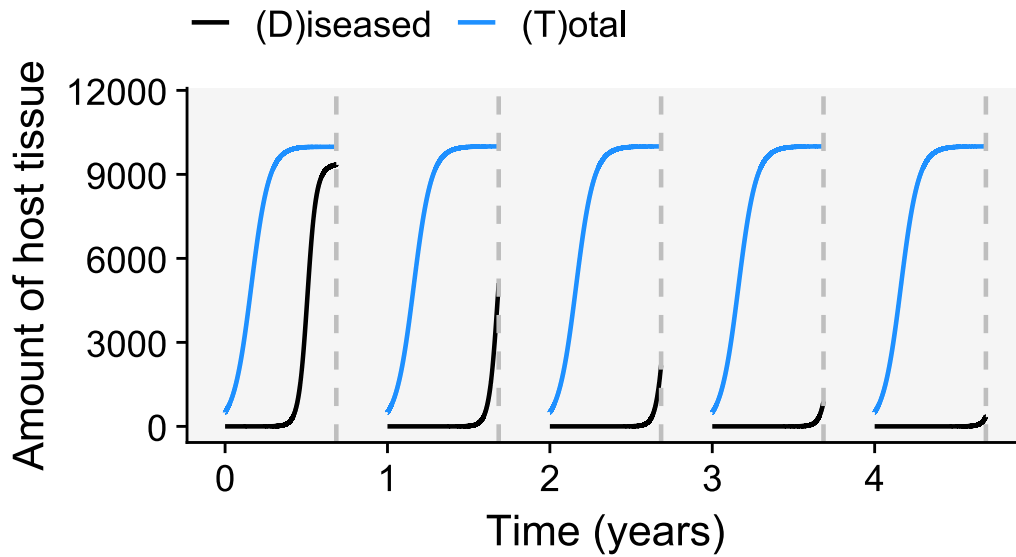
Epidemiological implications

The model including seasonality introduces new thresholds beyond the single-season basic reproduction number R_0 . In particular, even if there is a large epidemic within a season, if the carry over of inoculum between seasons is too small, the epidemic dies out.

This is illustrated below, in which the initial level of inoculum is much larger, but the production rate is much smaller (the code is omitted, which is identical apart from the values of the two parameters $P_1(0)$ and ϕ).

```
init.inoc <- 0.5
phi <- 1e-8
```

Multi-seasonal epidemic. Disease prc



Over successive seasons the amount of inoculum becomes smaller, and the epidemic dies out. This is because the smaller value of ϕ means that either less inoculum is produced within the season from a given amount of infectious tissue or that the inoculum is less good at surviving between seasons (these two mechanisms are not distinguishable within the current formulation of the model, although a more complex model could, of course, allow them to be disentangled).

Fitting to data

So far, compartmental models have been used to encode biological assumptions and to generate predictions about disease dynamics. In practice, however, epidemiological models

are often used in the opposite direction: observations of disease are used to estimate unknown model parameters.

To illustrate the basic idea, we consider a simple example based on the H-L-I-R model.

Suppose disease incidence has been measured through time during an epidemic and that all model parameters are known except the transmission rate, β .

The goal is to use the observations to estimate the value of β .

Generating a synthetic data set

To demonstrate the procedure, we first simulate an epidemic from the H-L-I-R model using a known value of $\beta = 0.00002 \text{ host}^{-1} \text{ day}^{-1}$. Note that the other parameters, as well as the initial conditions, are held fixed throughout; we are therefore only concentrating on estimating a single parameter.

We present a very simple treatment in which we assume all infected hosts – including latently infected hosts – are visible, and so would be scored as “diseased” in any experiment or field work. Of course, this could be challenged, and would definitely depend on the pathosystem of interest.

The key output of the model for fitting is therefore a set of values of $D = L + I + R$ at particular times, t .

Before fitting, we add a small amount of artificial noise to these “data”. For illustrative purposes, we add Gaussian noise whose variance is scaled to resemble that of a binomial process.

In the computational work hereafter, these observations are treated as if they were experimental data, and are used to estimate the value of the infection rate, β .

```
library(deSolve)

find.dpc.hlir.known.beta <- function(this.beta,
                                     time.step = 20)
{
  this.hlir.params <- c(beta = this.beta,
                       gamma = 0.2,
                       mu = 0.1)

  # Specify initial conditions
  this.hlir.init <- c( H = 9999,
                     L = 1,
                     I = 0,
                     R = 0)

  # Specify range of times for which a solution is required
  this.hlir.time <- seq(0, 250, by = time.step)

  # Solve the model
  this.hlir.sol <- ode(y = this.hlir.init,
                     times = this.hlir.time,
                     func = hlir.model,
                     parms = this.hlir.params)

  # Cast results to a R data frame
  this.hlir.sol.df <- data.frame(this.hlir.sol)
```

```

# Extract D = L + I + R and t
dpc.data <- data.frame(time = this.hilir.sol.df$time,
                      D = this.hilir.sol.df$L + this.hilir.sol.df$I + this.hilir.sol.df$R)

return(dpc.data)
}

# seed the random number generator for replicability
set.seed(07072026)

true.beta <- 0.00002
this.N <- 10000

orig.dpc <- find.dpc.hilir.known.beta(true.beta)

p.disease <- orig.dpc$D / this.N
noise.sd <- sqrt(this.N * p.disease * (1 - p.disease))

orig.dpc$D.obs <- orig.dpc$D + 25 * noise.sd * rnorm(length(orig.dpc$D))
# truncate any out of range observations
orig.dpc$D.obs <- pmax(0, pmin(this.N, orig.dpc$D.obs))

```

Defining a measure of goodness of fit

To estimate β we need a measure of agreement between the model (for different values of β) and the observations.

A simple choice is the sum of squared errors,

$$S(\beta) = \sum_i (D_{obs}(t_i) - D_{pred}(t_i; \beta))^2 \quad (12.20)$$

where $D_{obs}(t_i)$ is the observed amount of disease at time t_i , and $D_{pred}(t_i; \beta)$ is the corresponding model prediction for a particular value of β .

The value of β that minimises $S(\beta)$ is the best-fitting estimate of the transmission rate

```

find.ss.beta <- function(beta, data)
{
  this.dpc <- find.dpc.hilir.known.beta(beta)
  SS <- sum((this.dpc$D - data$D.obs)^2)
  return(SS)
}

```

Visualising the objective function

Before performing numerical optimisation, it is sensible to calculate $SS(\beta)$ over a range of possible transmission rates, and to plot the results. Note that before plotting the y-value (the sum of squares) is divided by one million.

```

library(ggplot2)
library(r4pde)

all.beta <- seq(0.5 * true.beta,
              1.5 * true.beta,
              length.out = 101)
all.ss <- numeric(length(all.beta))
for(i in 1:length(all.beta))

```



```

{
  all.SS[i] <- find.ss.beta(all.beta[i],orig.dpc)
}
all.SS <- all.SS / 1000000

min_idx <- which.min(all.SS)
fitted_beta <- all.beta[min_idx]
min_ss <- all.SS[min_idx]

ss_df <- data.frame(
  beta = all.beta,
  ss = all.SS
)

p <- ggplot(ss_df, aes(x = beta, y = ss)) +
  geom_line(aes(color = "Sum of squares"), linewidth = 0.8) +
  geom_vline(aes(xintercept = true.beta, color = "True beta"), linewidth = 0.8) +
  geom_vline(aes(xintercept = fitted_beta, color = "Fitted beta"), linetype = "solid", linewidth = 0.8) +
  geom_point(data = data.frame(beta = fitted_beta, ss = min_ss),
    aes(x = beta, y = ss, color = "Fitted beta"), size = 3) +
  scale_color_manual(
    values = c(
      "Sum of squares" = "black",
      "True beta" = "blue",
      "Fitted beta" = "red"
    )
  ) +
  labs(
    x = "Infection rate",
    y = "Sum of squares (millions)",
    title = "Sum of squares as a function of infection rate",
    color = NULL
  ) +
  theme_r4pde() +
  theme(legend.position = c(0.05, 0.25)) +
  guides(color = guide_legend(override.aes = list(linetype = "solid")))

print(p)

```

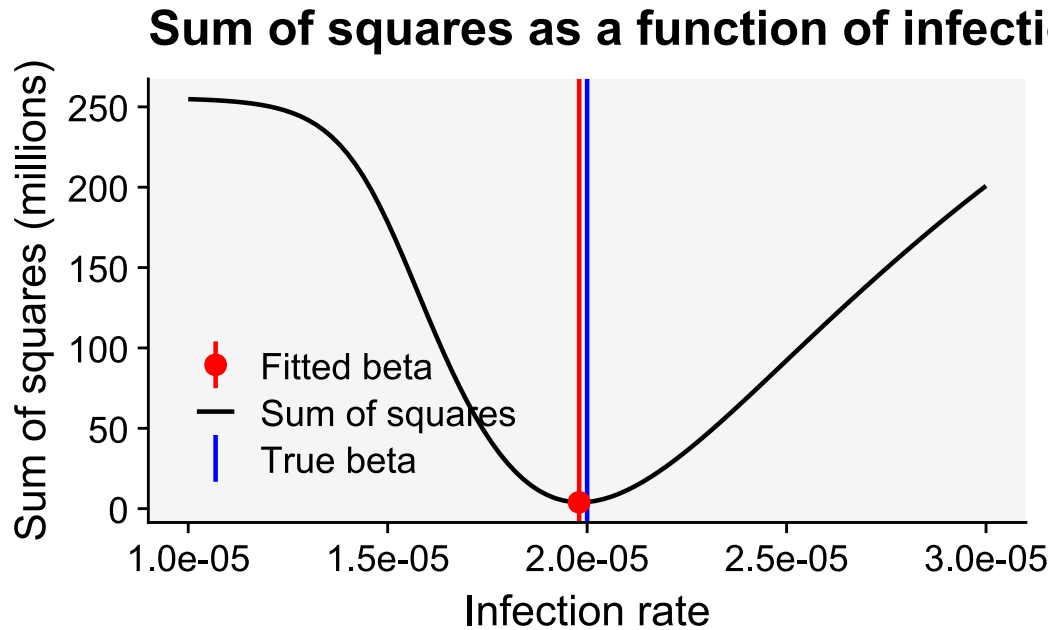


Figure 12.7: Sum of squares as a function of the infection rate, showing the true and the fitted values of the transmission rate

The sum of squares has a clear minimum. This is the value of β which has the closest match between model and observations.

The value has been obtained above by a one-way scan as

```
print(paste("Estimated beta (one way scan) =", all.beta[which.min(all.SS)]))
```

```
[1] "Estimated beta (one way scan) = 1.98e-05"
```

Numerical optimisation

Although the minimum can be identified visually, optimisation algorithms provide a more efficient and general approach. In the code below, this is done using the `optim()` function in R. This requires an initial guess at the optimal value of β ; it is set to be a random value within 50% of the true value.

```
fit <- optim(par = true.beta * runif(1,min=0.5,max=1.5),
  fn = find.ss.beta,
  data = orig.dpc,
  method = "L-BFGS-B",
  lower = 0,
  control = list(parscale = 1e-5)) # note need last argument to
                                # avoid optimiser thinking it has
                                # converged when it has not
                                # (due to large values of SS)

beta.hat <- fit$par

print(paste("Estimated beta (optim) =", beta.hat))
```

```
[1] "Estimated beta (optim) = 1.98176029764102e-05"
```

The `parscale` argument rescales the optimisation problem so that changes in β of order 10^{-5} are treated as numerically meaningful by the optimiser.

The estimate of β is very close to the true value used to generate the synthetic data. In a real application, however, the true parameter value would of course be unknown.

Interpretation

To assess the quality of the fit, the fitted model predictions can be plotted against the observations (see below).

```
library(ggplot2)
library(r4pde)

fitted.dpc <- find.dpc.hlr.known.beta(beta.hat,
                                     time.step=1)

# recalculate the dpc for the original data set
# to make the graph appear smoother
orig.beta.dpc <- find.dpc.hlr.known.beta(true.beta,
                                         time.step=1)

p <- ggplot() +
  # Observations (points)
  geom_point(data = orig.dpc, aes(x = time, y = D.obs, color = "Observations"), size = 2) +
  # Fitted model (solid line)
  geom_line(data = fitted.dpc, aes(x = time, y = D, color = "Fitted model"), linewidth = 0.8) +
  # Original parameters (dashed line)
  geom_line(data = orig.beta.dpc, aes(x = time, y = D, color = "Original parameters"), linetype = "dotted",
            linewidth = 0.8) +
  scale_color_manual(
    values = c(
      "Observations" = "red",
      "Fitted model" = "black",
      "Original parameters" = "blue"
    )
  ) +
  labs(
    x = "Time (days)",
    y = "Number of diseased hosts (L + I + R)",
    title = "Results of fitting the model",
    color = NULL
  ) +
  xlim(0, 250) +
  ylim(0, 10000) +
  theme_r4pde() +
  theme(legend.position = c(0.05, 0.9))

print(p)
```

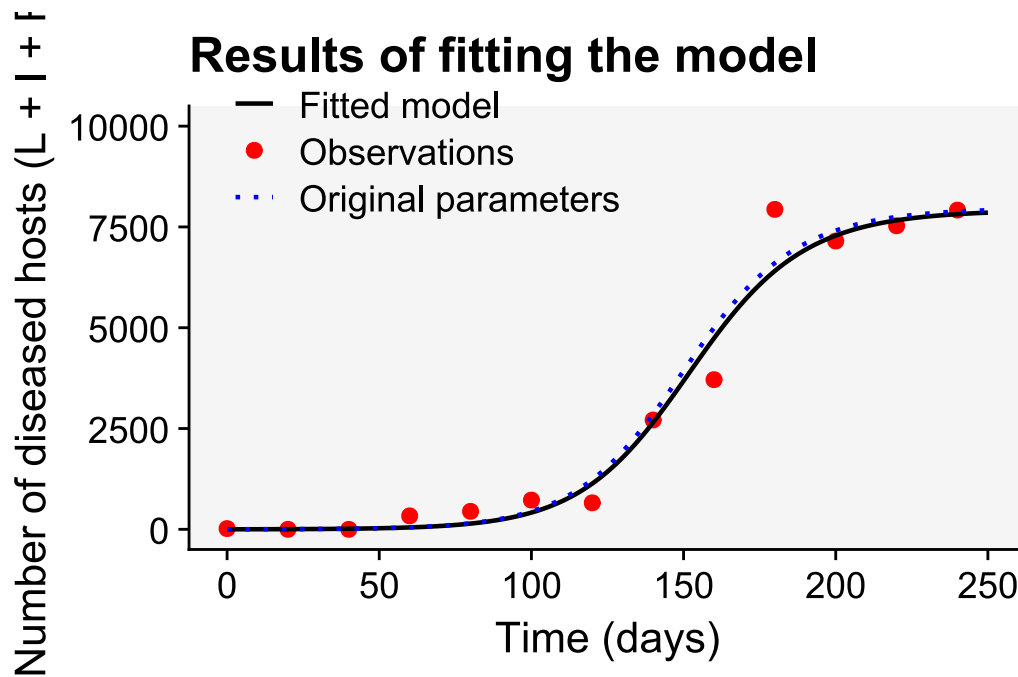


Figure 12.8: Observations (points), the fitted model (solid line) and the model with the original parameter value (dotted line)

The fitted model performs very well.

Summary

This simple example illustrates a general feature of epidemiological modelling: model parameters can be estimated by comparing the degree of concordance between observations and predictions. In practice we usually want confidence intervals rather than only point estimates, but the above shows the basic idea of how parameters can be estimated from disease progress data.

Although only a single parameter was estimated here, the same principles extend to more realistic applications. In practice, several parameters may be unknown simultaneously, data may be noisy or incomplete, and different combinations of parameter values may produce very similar epidemic trajectories. This final issue is known as parameter identifiability (Cunniffe et al., 2024). Consequently, determining which parameters can be reliably estimated from a given dataset is often an important modelling challenge in its own right.

More sophisticated approaches, including likelihood-based and Bayesian methods, are widely used in modern epidemiology. Nevertheless, the underlying principle remains the same: parameters are estimated by identifying the model that best explains the observed data.

Further extensions

The H–L–I–R framework provides a useful and biologically interpretable starting point for modelling plant disease epidemics. However, the assumptions underpinning the model are often simplifications.

Above, we have shown how some of these assumptions can be relaxed. However, many applications require extensions that relax one or more of these assumptions. Important extensions include:

- **Multiple strains**, allowing investigation of competition, coexistence and evolutionary change. Examples of models in the literature tracking multiple pathogen strains include Elderfield et al. (2018), Allen et al. (2019), Hamelin et al. (2019) and Taylor & Cunliffe (2023).
- **Multiple hosts**, allowing investigation of pathosystems in which more than one host or variety of host can become infected. Examples include Bussell & Cunliffe (2022), Murray-Watson & Cunliffe (2023) and Del Ponte et al. (2026).
- **Vector-borne transmission**, where insect vectors must be modelled alongside host populations. Examples include Donnelly et al. (2019), Cunliffe et al. (2021), Zaffaroni et al. (2021) and Falla & Cunliffe (2026).
- **Spatial structure**, ranging from individual based models to patch-based models to explicit representations of pathogen dispersal across landscapes. Examples include Cunliffe et al. (2015), Cunliffe et al. (2016), Hyatt-Twynam et al. (2017) and Ellis et al. (2025).
- **Stochastic dynamics**, which capture chance events during invasion, extinction and disease persistence. Examples include Thompson et al. (2020), Fabre et al. (2021) and Russell & Cunliffe (2025).
- **Inference from data**, where compartmental models are linked to observations to estimate parameters and quantify uncertainty. Examples include Gibson & Austin (1996), Adrakey et al. (2017), Thompson et al. (2018) and Best & Cunliffe (2026).

Although these extensions increase biological realism, they also introduce additional parameters, data requirements and computational complexity. Nevertheless, they retain the central idea introduced in this chapter: representing disease processes through biologically meaningful compartments and flows between them.

Summary

Compartmental models are probably best viewed as a flexible framework rather than a single model. A major strength of the framework is that additional biological complexity can often be incorporated simply by introducing new compartments or modifying transition rates. Starting from simple formulations, additional biological realism can be introduced in targeted ways to reflect specific epidemiological settings. The key challenge is not to maximise complexity, but to identify the simplest model that captures the dominant processes governing the epidemic. This principle underlies much of modern infectious disease modelling, irrespective of whether the pathogens infect plants, humans or animals. Throughout, the same framework serves three complementary purposes: representing biological mechanisms, understanding epidemic behaviour and extracting information from data.

Appendix. Relationship to population dynamics models

It is useful to show how simpler population dynamics models (as covered in the chapter on [population models](#)) arise as limiting cases of the compartmental modelling framework.

We start with the H-L-I-R-(P) model (without host growth).

Assume that inoculum does not decay, i.e., $\omega = 0$. This means the density of inoculum is fixed at $P(0)$ for the whole epidemic, and the equation for dP/dt can be omitted.

Introduce the symbol $\epsilon = \alpha P(0)$ to represent the fixed rate of primary infection (per healthy host) throughout the epidemic.

The H-L-I-R-(P) model then becomes

$$\begin{aligned}\frac{dH}{dt} &= -(\epsilon + \beta I)H, \\ \frac{dL}{dt} &= (\epsilon + \beta I)H - \gamma L, \\ \frac{dI}{dt} &= \gamma L - \mu I, \\ \frac{dR}{dt} &= \mu I.\end{aligned}\tag{12.21}$$

Assume that the latent period is infinitesimally short (i.e., $\gamma \rightarrow \infty$) and that infectious tissue does not lose infectivity (i.e., $\mu = 0$). This simplifies the dynamics

$$\begin{aligned}\frac{dH}{dt} &= -(\epsilon + \beta I)H, \\ \frac{dI}{dt} &= (\epsilon + \beta I)H.\end{aligned}\tag{12.22}$$

But then, since $H + I = N$, the model reduces down to a single equation for the number of infected plants $\frac{dI}{dt} = (\epsilon + \beta I)(N - I)$. As a historical aside, note this form of model was introduced by [Brasset & Gilligan \(1988\)](#) as a “differential equation that combines the properties of the monomolecular and logistic models”, and used to understand take-all disease in wheat. Here we have seen how it can be derived from the H-L-I-R model.

Recall the simpler population dynamics models are normally phrased in terms of the proportion of diseased plants/tissue. As described near the start of this chapter, this can be done here by considering $y = I/N$, which leads to $\frac{dy}{dt} = (\epsilon + \beta N y)(1 - y)$.

Logistic model

The logistic model is

$$\frac{dy}{dt} = r_L y(1 - y),\tag{12.23}$$

in which r_L is the rate parameter. The solution is

$$y = \frac{1}{1 + \left(\frac{1-y_0}{y_0}\right) \exp(-r_L t)},\tag{12.24}$$

where y_0 is the initial proportion of disease.

This is equivalent to the H-L-I-R-(P) model in the limit $\epsilon \rightarrow 0$, i.e., when there is no primary infection, and with the parameter identification $\beta N = r_L$.

Monomolecular model

The monomolecular model is

$$\frac{dy}{dt} = r_M(1 - y),\tag{12.25}$$

in which r_M is the rate parameter. The solution is

$$y = 1 - (1 - y_0) \exp(-r_M t)\tag{12.26}$$

where y_0 is the initial proportion of disease.

This is equivalent to the H-L-I-R-(P) model in the limit $\beta \rightarrow 0$, i.e., when there is no secondary infection, and with the parameter identification $\epsilon = r_M$.

Exponential model

The exponential model is

$$\frac{dy}{dt} = r_E y, \quad (12.27)$$

in which r_E is the rate parameter. The solution is

$$y = y_0 \exp(r_E t) \quad (12.28)$$

where y_0 is the initial proportion of disease.

This is equivalent to the H-L-I-R-(P) model in the limit $\epsilon \rightarrow 0$ and $y \approx 0$, i.e., the exponential model is recovered when there is no primary infection and when y remains small, with the parameter identification $\beta N = r_E$

$$\frac{dy}{dt} \approx \beta N y. \quad (12.29)$$

Gompertz model

A concordance between process-based models and the Gompertz model can be made if we move to a more complex Kermack-McKendrick type model in which the infection rate depends on the time-since-infection (Cunniffe et al., 2012; Segarra et al., 2001), at least approximately. However, this is really quite advanced and requires going well beyond what can be covered in this chapter, so we do not pursue this here.

PART III

Spatial analysis

13. Spatial gradients

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 following sections, we explore examples of actual disease gradients measured in the field, each exhibiting its own unique pattern.

Our first example, from the study by [Mundt et al. \(1999\)](#), sought to measure the dispersal potential of the bacterium *Xanthomonas oryzae* pv. *oryzae*, which causes bacterial blight of 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())

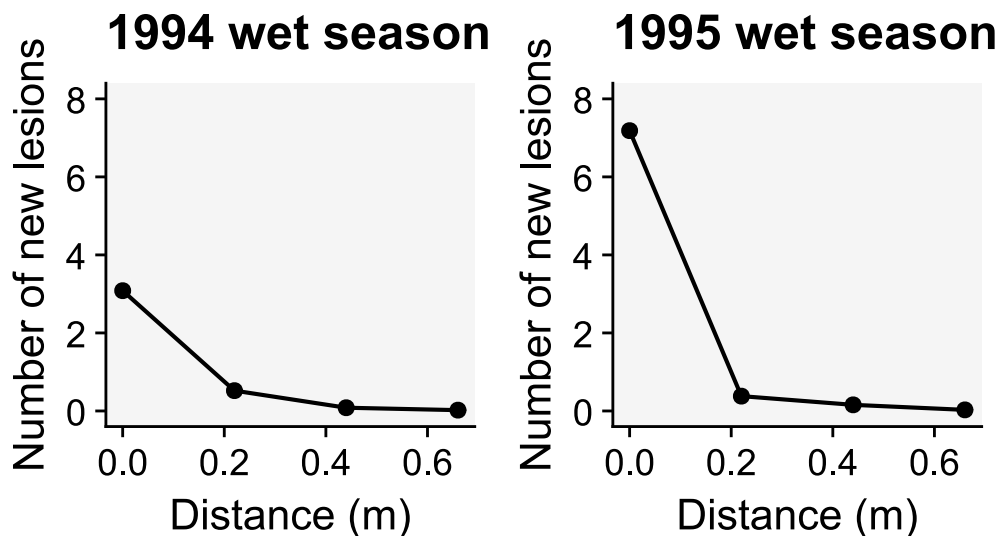
xo <-
  tibble::tribble(
    ~d, ~y4, ~y5,
    0, 3.083, 7.185,
    0.22, 0.521, 0.38,
    0.44, 0.083, 0.157,
    0.66, 0.021, 0.028
  )

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)")
```



Source: Mundt et al. (1999)

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

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 by Sackett & 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,
```

```

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
)

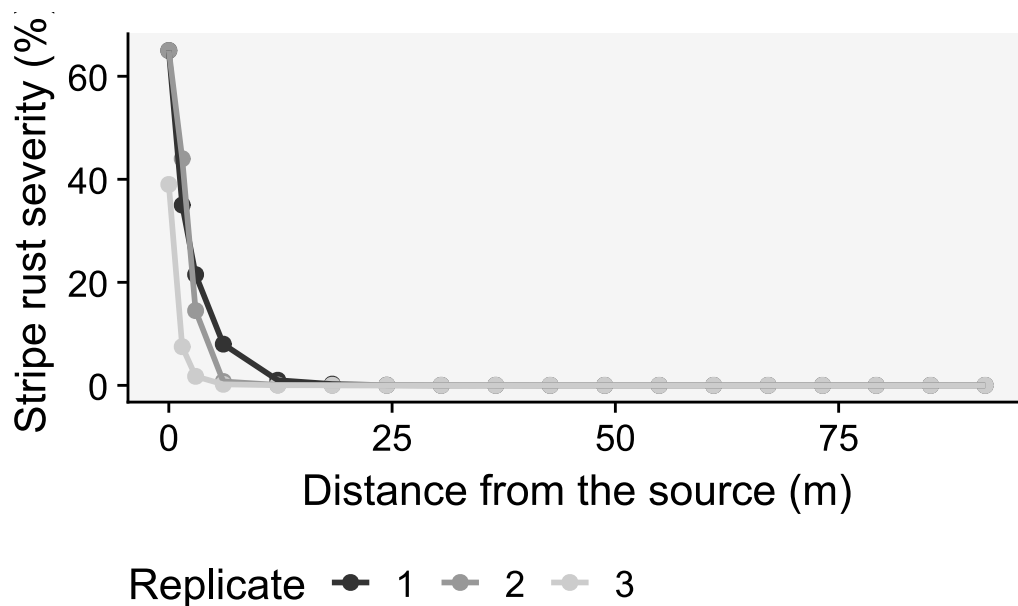
```

```

library(tidyverse)
library(ggthemes)

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(linewidth = 1)+
  scale_color_grey()+
  labs(x = "Distance from the source (m)",
       y = "Stripe rust severity (%)",
       color = "Replicate",
       caption = "source: Sackett et al. (2005)")

```



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 closer 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 on two dates, 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 tibble and the plot produced using *ggplot2*.

```
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)")
```

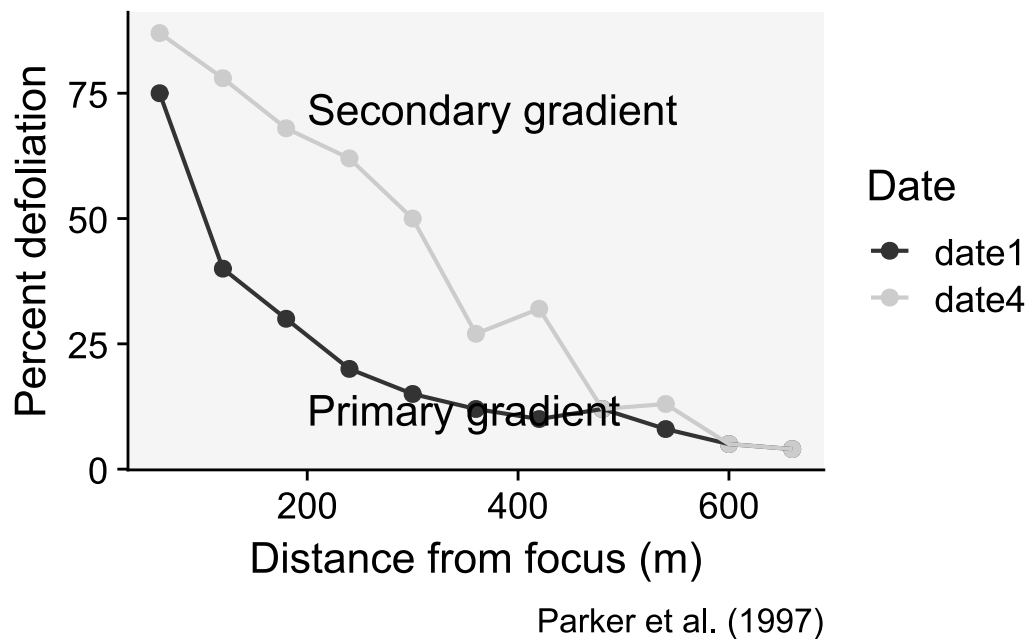


Figure 13.3: Primary and secondary gradients of defoliation due to Septoria leaf spot on tomato

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** (plural: foci), 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 = "#339966")+
  annotate("segment", linewidth = 2, x = 1, xend = 1, y = 9.5, yend = 2, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 3, xend = 3, y = 9.5, yend = 2, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 5, xend = 5, y = 9.5, yend = 2, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 7, xend = 7, y = 9.5, yend = 2, arrow = arrow())+
  annotate("segment", linewidth = 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 = "#339966")+
  annotate("segment", linewidth = 2, x = 1, xend = 1, y = 8, yend = 2, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 3, xend = 3, y = 8, yend = 2, arrow = arrow())
```

```

annotate("segment", linewidth = 2, x = 5, xend = 5, y = 8, yend = 2, arrow = arrow())+
annotate("segment", linewidth = 2, x = 7, xend = 7, y = 8, yend = 2, arrow = arrow())+
annotate("segment", linewidth = 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", linewidth = 2, x = 5, xend = 10, y = 5, yend = 10, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 5, xend = 10, y = 5, yend = 5, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 5, xend = 10, y = 5, yend = 0, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 5, xend = 0, y = 5, yend = 0, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 5, xend = 0, y = 5, yend = 5, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 5, xend = 0, y = 5, yend = 10, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 5, xend = 5, y = 5, yend = 10, arrow = arrow())+
  annotate("segment", linewidth = 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 = "#339966" )+
  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", linewidth = 2, x = 0, xend = 10, y = 10, yend = 0, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 0, xend = 6.6, y = 10, yend = 0, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 0, xend = 3.3, y = 10, yend = 0, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 0, xend = 0, y = 10, yend = 0, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 0, xend = 10, y = 10, yend = 3.3, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 0, xend = 10, y = 10, yend = 6.6, arrow = arrow())+
  annotate("segment", linewidth = 2, x = 0, xend = 10, y = 10, yend = 10, arrow = arrow())+
  annotate("rect", xmin = 0, xmax = 1, ymin = 9, ymax = 10, color = "black", fill = "#339966")+
  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")

```

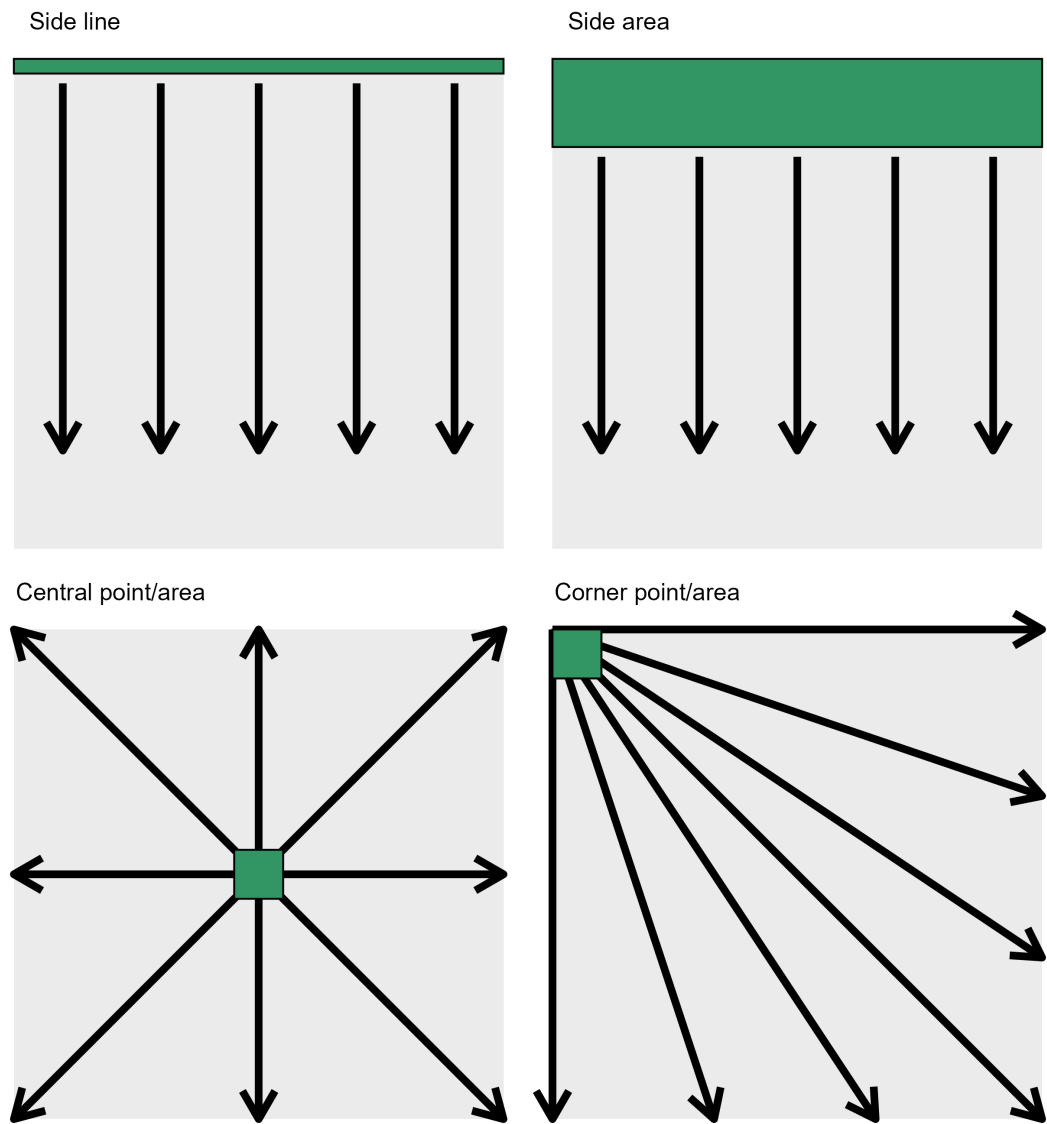


Figure 13.4: Example of location and size of inoculum sources for the study of disease gradients

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 in [this tutorial](#).

Exponential model

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

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

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

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

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 us 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)"
  )
```

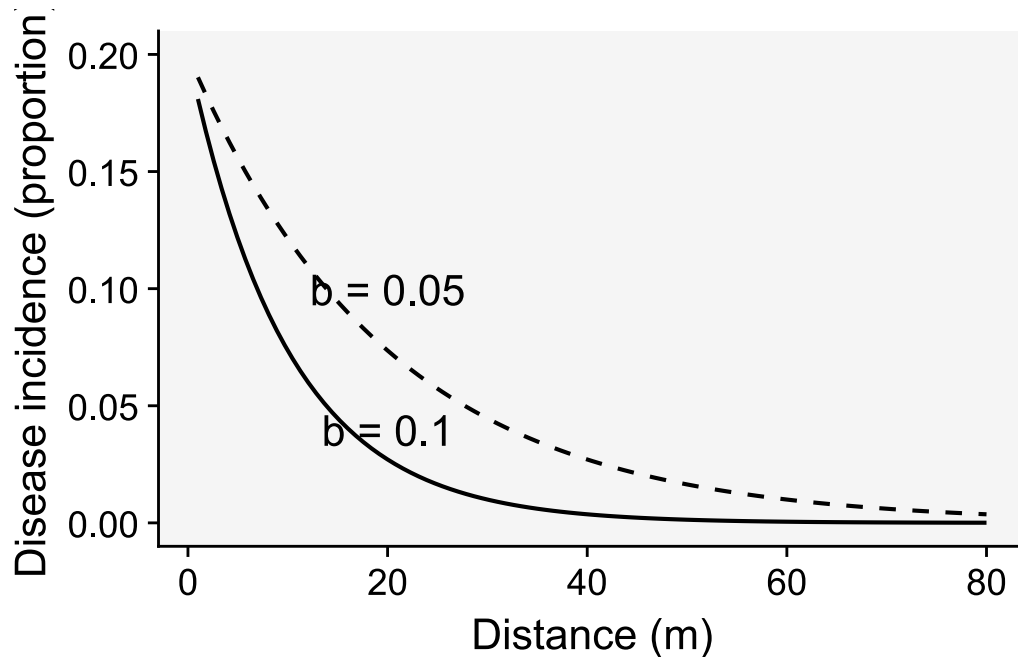


Figure 14.1: Exponential curves describing plant disease gradients

Power law model

Also known as the modified Gregory's model (Gregory was a pioneer in the use of 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 x approaches zero, Y becomes 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 according 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 more appropriate to record the distance in cm than in m or km.

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

$$Y = a_P (x + C)^{-b_P}$$

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

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

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

```
dat2 |>
  ggplot(aes(x, y)) +
  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 = 0.7") +
  annotate("text", x = 20, y = 0.06, label = "b = 0.5") +
  labs(x = "Distance (m)", y = "Disease incidence")
```

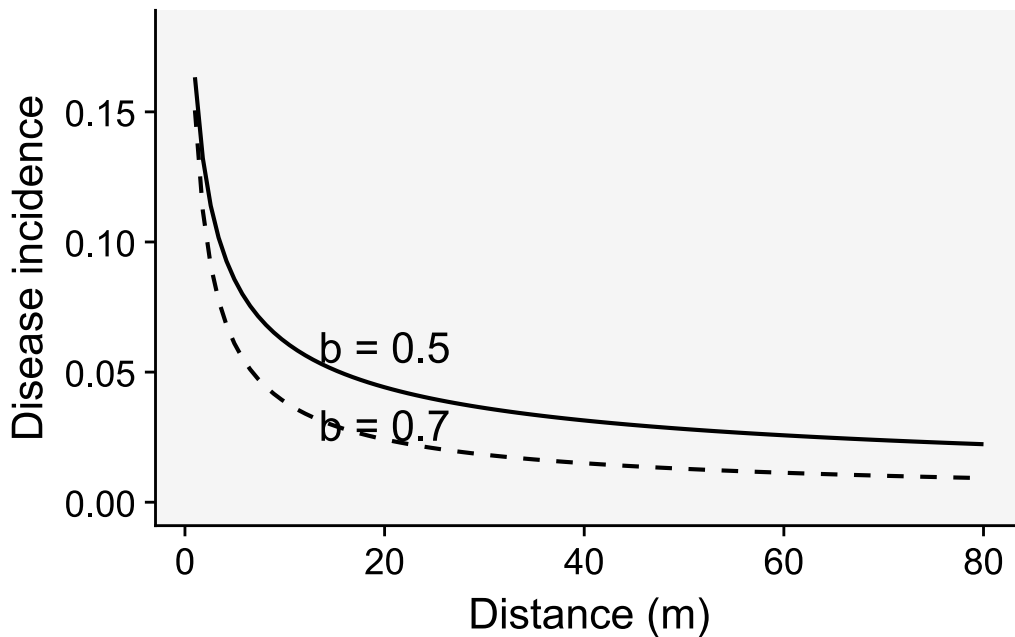


Figure 14.2: Power law (modified) curves describing plant disease gradients

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.

Linearization of the models

Transformations of y

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

$$y^* = a^* + bx,$$

where the asterisk in a indicates 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. The linearized forms of the two most common gradient models are:

$$\ln(y) = \ln(a_E) - b_E \cdot x$$

$$\ln(y) = \ln(a_P) - b_P \cdot \ln(x + C)$$

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 $\ln(y)$.

```
a1 <- 0.2 # y at zero distance for gradient 1
a2 <- 0.2 # y at zero distance for gradient 2
b1 <- 0.1 # decline rate for gradient 1
b2 <- 0.05 # decline rate for gradient 2

ggplot(data.frame(x = c(0, 80)), aes(x)) +
  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 = "Distance (m)", y = "log of disease incidence")
```

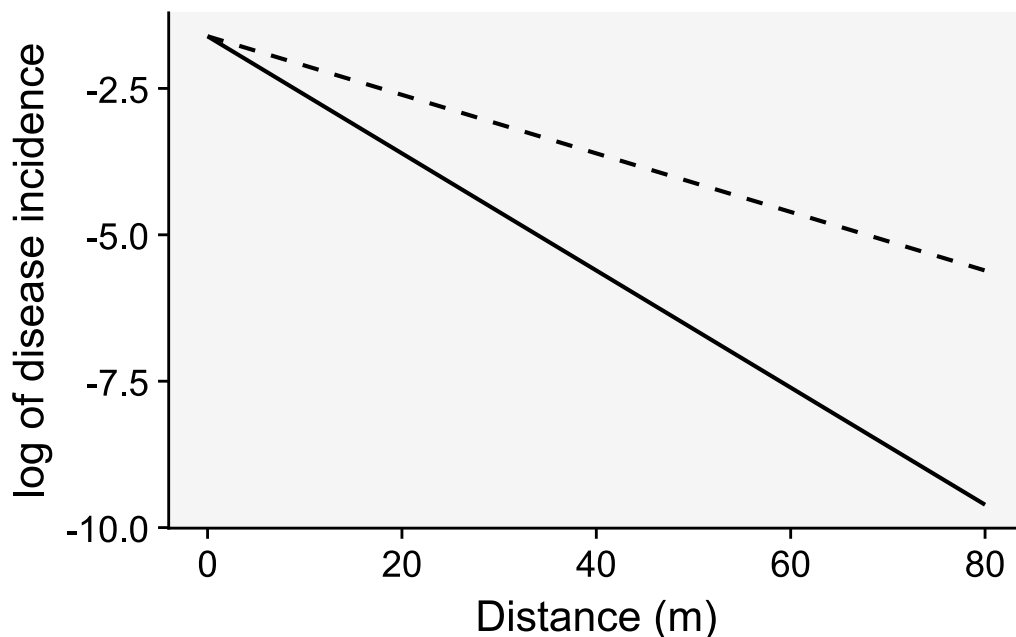


Figure 14.3: Linearization of the exponential model describing plant disease gradients

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

```
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

# the x axis is the transformed distance, log(x + C)
ggplot(data.frame(u = c(log(C), log(80 + C))), aes(u)) +
  theme_r4pde() +
  stat_function(fun = function(u) log(a1) - b1 * u, linetype = 1) +
  stat_function(fun = function(u) log(a2) - b2 * u, linetype = 2) +
  labs(
    title = "Modified Power Law",
    x = "log of (distance + C)",
    y = "log of disease incidence"
  )
)
```

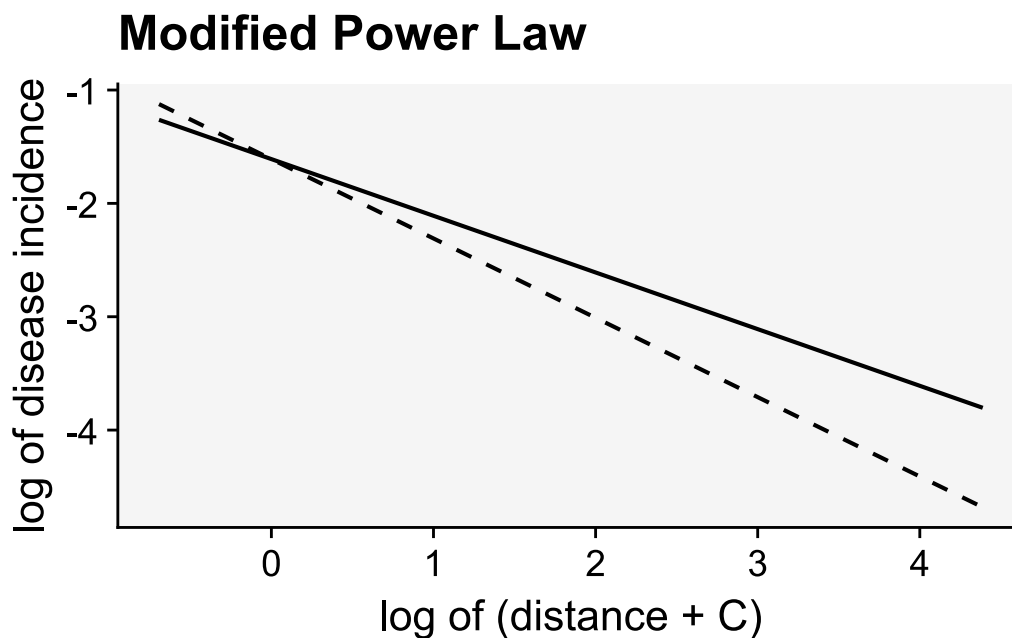


Figure 14.4: Linearization of the modified power law curves describing plant disease gradients

Interactive application

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

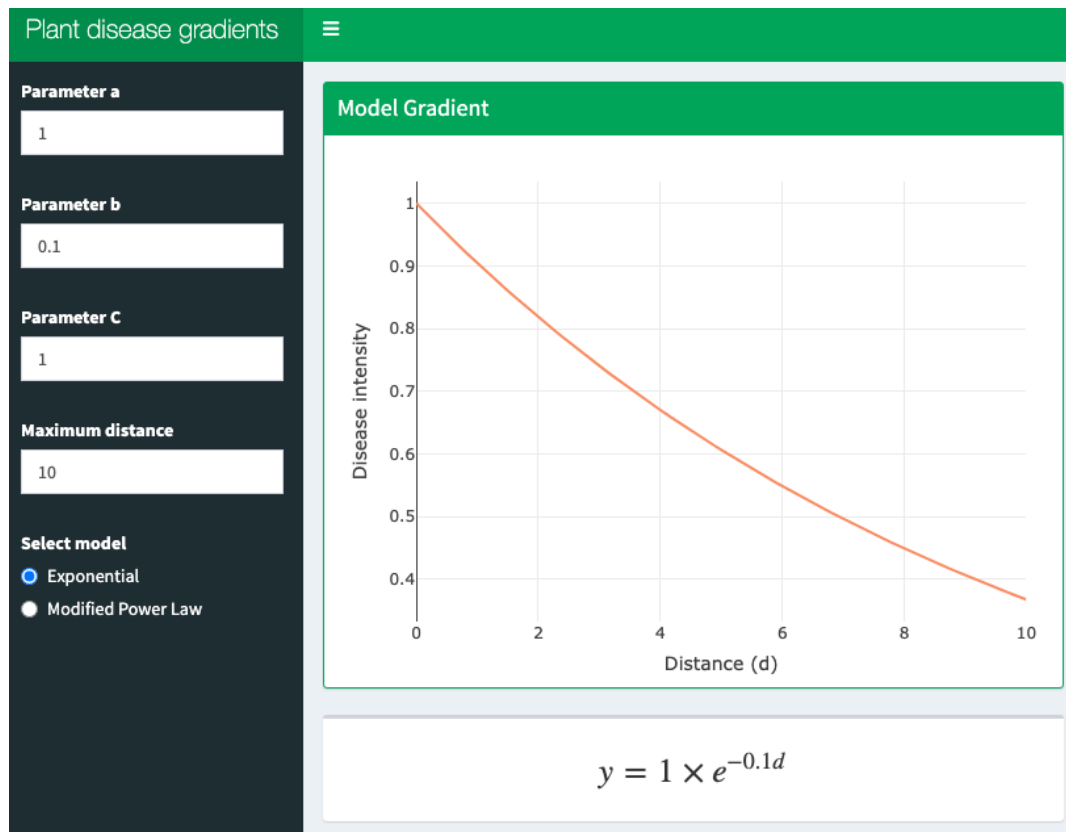


Figure 14.5: Screenshot of the application to visualize the spatial disease gradient models by varying the model's parameters

15. Fitting gradient models

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

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

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)")
```

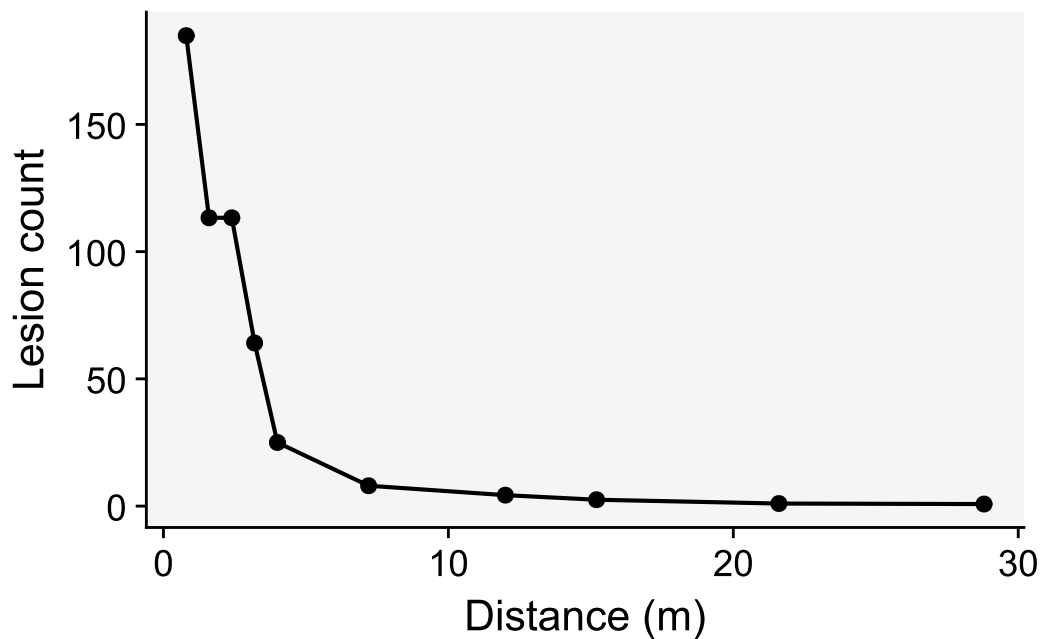


Figure 15.1: Hypothetical gradient of lesion count over distances from the inoculum source

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.

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
R²	0.88
Adj. R²	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 in the 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)")
```

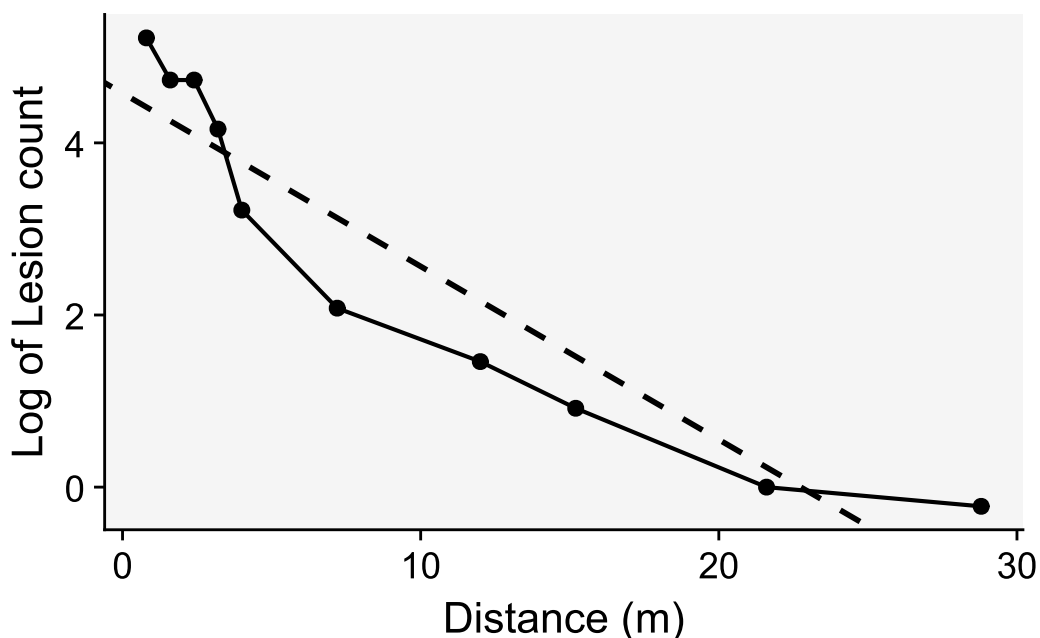


Figure 15.2: Fit of the exponential model to the log of lesion count over distances from the inoculum source

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 in the residuals surrounding the regression fit line, suggesting that the model may not capture all the underlying structures in the data.

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 va
(Intercept)	5.56	0.25 22.66
log(x)	-1.70	0.12 -14.26
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")
```

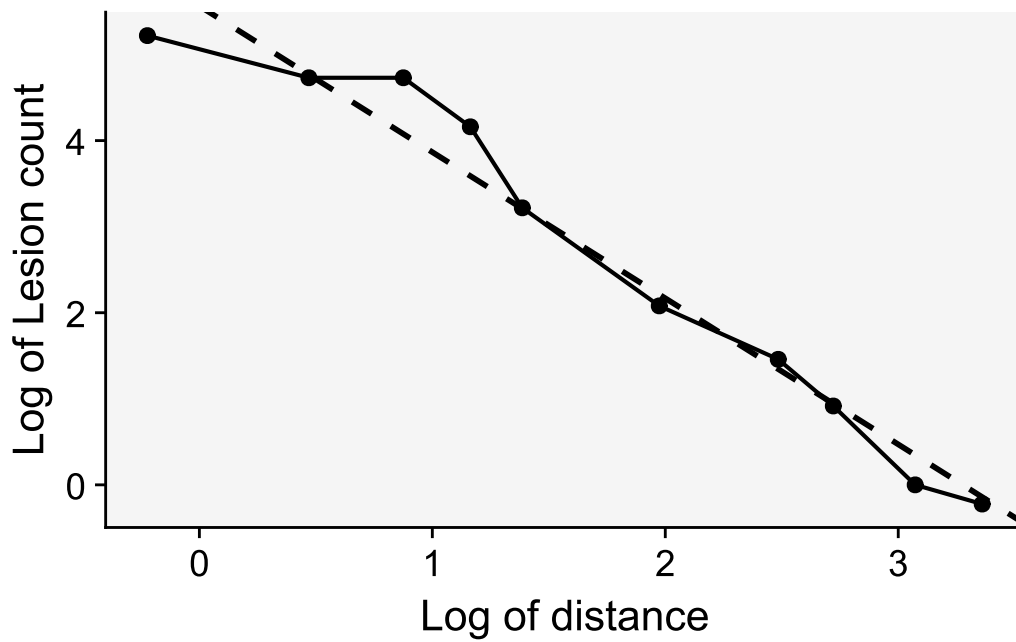


Figure 15.3: Fit of the power law model to the log of lesion count over log of the distance from the inoculum source

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)
```

Observations	10
Dependent variable	log(Y)
Type	OLS linear regression
F(1,8)	302.16
R²	0.97
Adj. R²	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

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

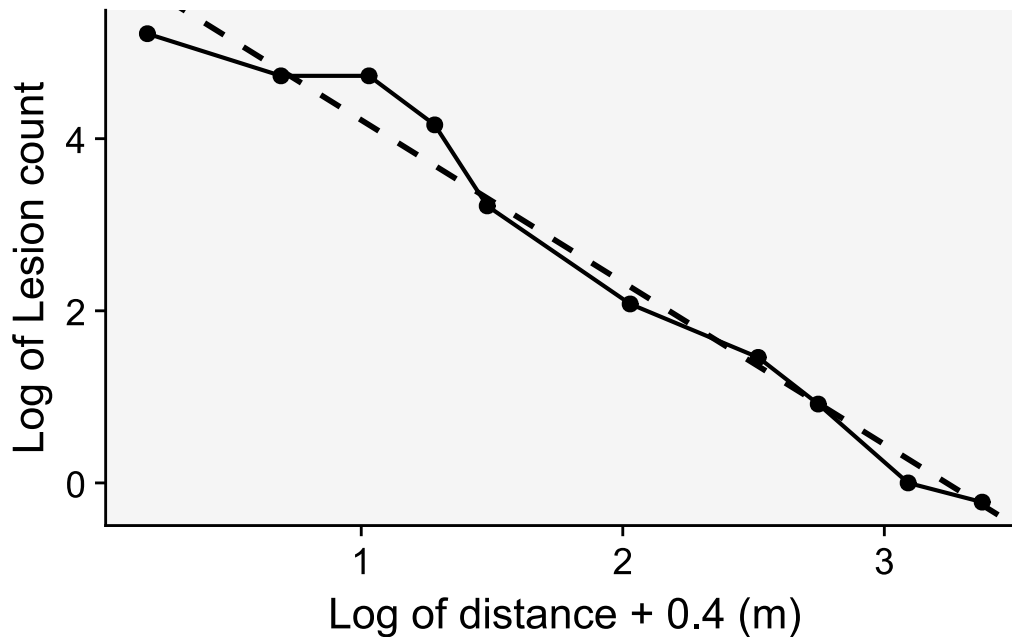
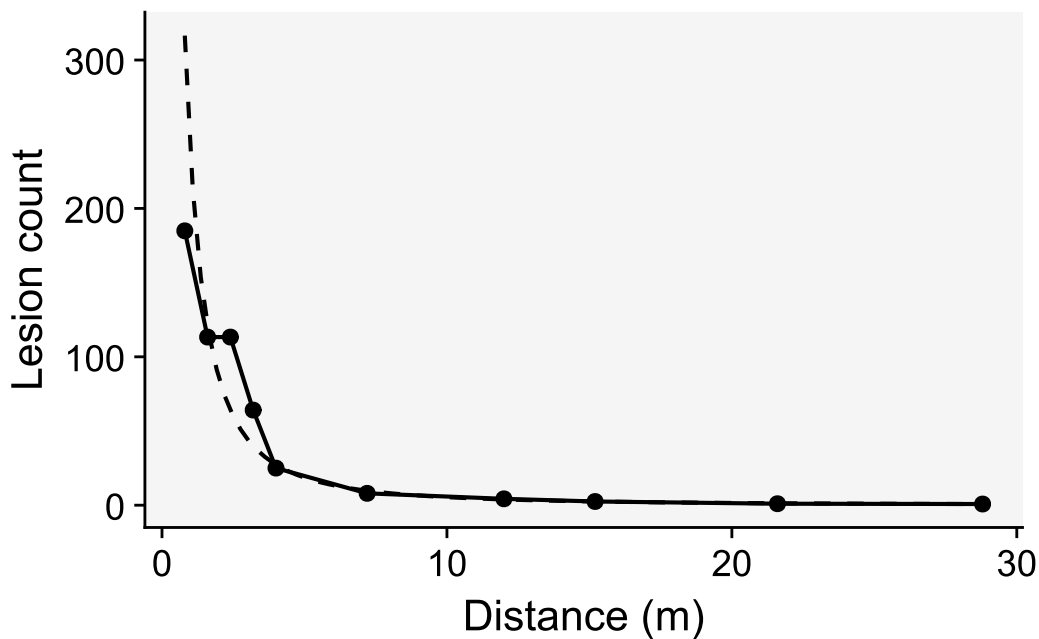


Figure 15.4: Fit of the modified power law model to the log of lesion count over log + 0.4 of the distances from the inoculum source

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_pm)[[2]]), linetype
= 2) +
  labs(y = "Lesion count",
       x = "Distance (m)")
```



Fitting all models with `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.

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

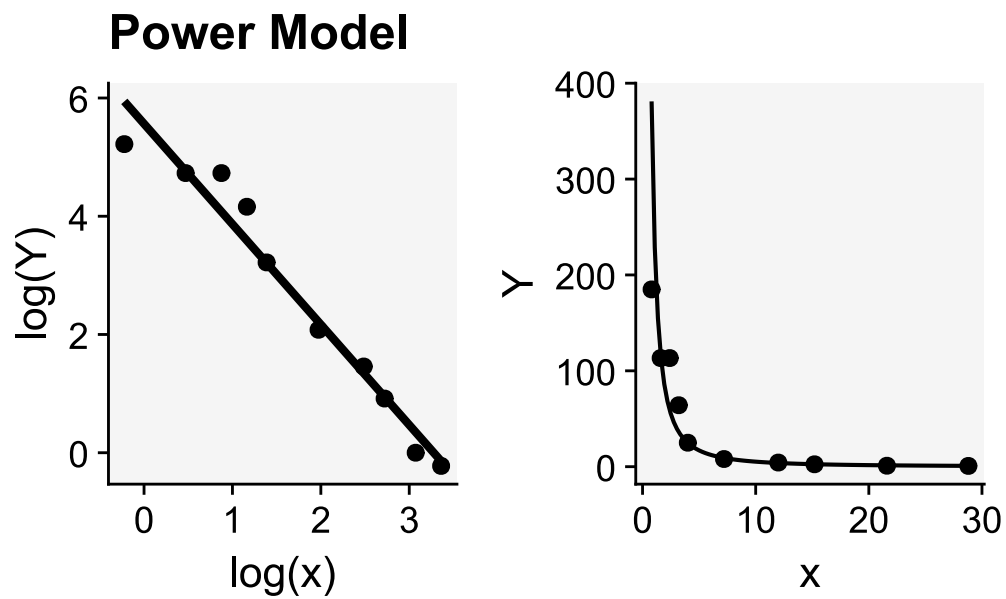
fit1 <- fit_gradients(grad1, C = 0.4)

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

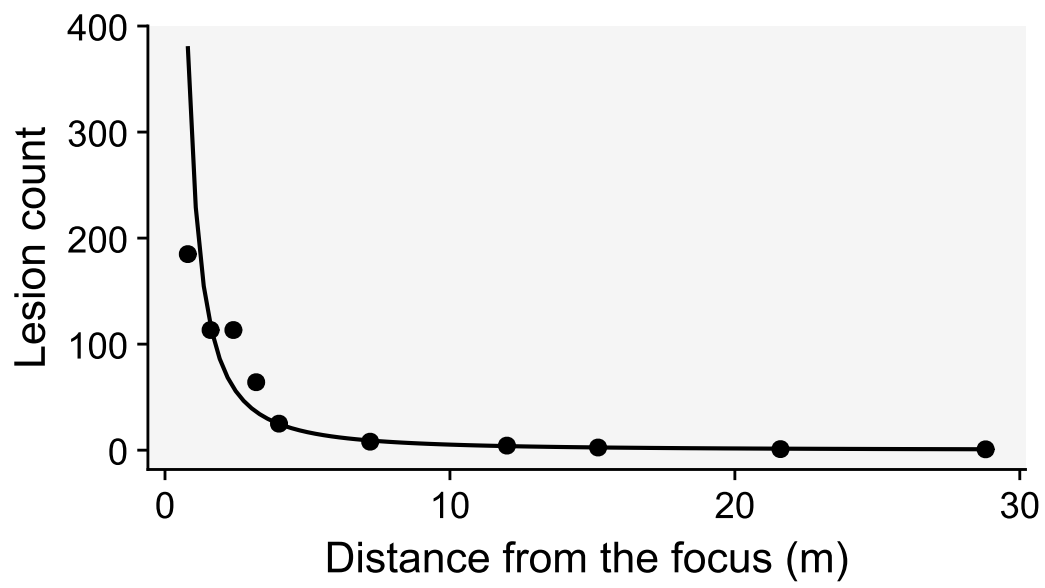
	a.(Intercept)	se_a	sig_a	b.x	se_b	sig_b	a_back.(Intercept)	R2
Power	5.564	0.246	**	-1.698	0.119	**	260.864	0.962
Modified_Power	6.101	0.228	**	-1.884	0.108	**	446.304	0.974

```
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

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., 2007c). 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.

```
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
```

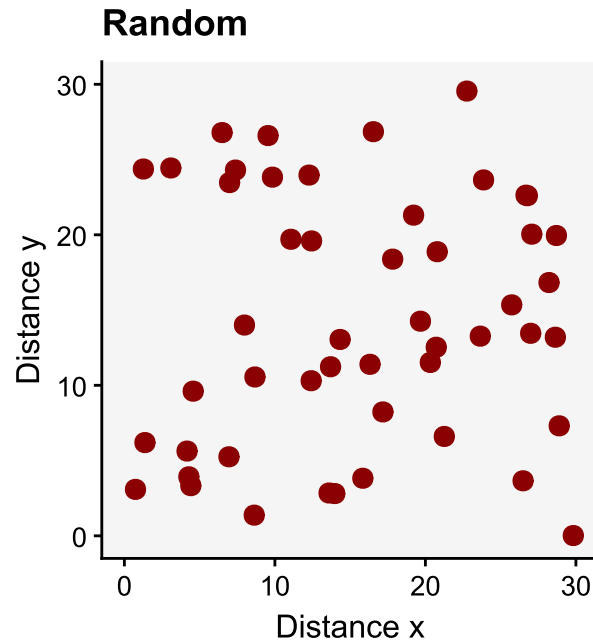


Figure 16.1: Random pattern of a plant disease epidemic

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
```

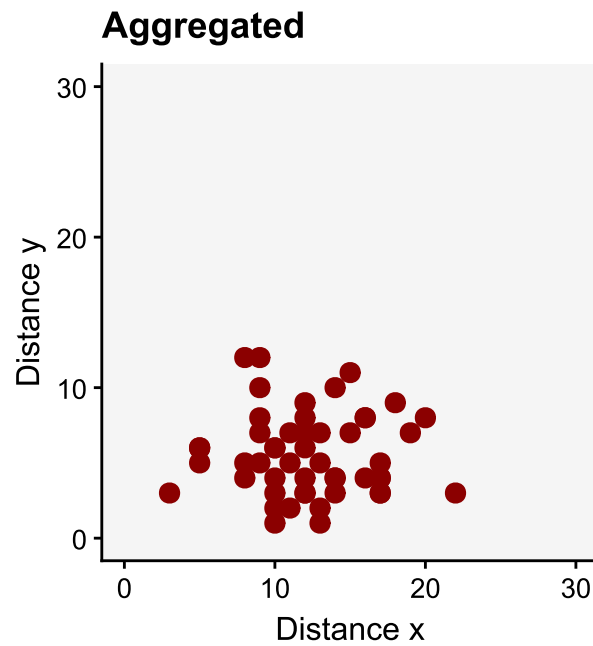



Figure 16.2: Aggregated pattern of a plant disease epidemic

A rare pattern found in nature is the regular pattern, but it may be generated artificially by humans when conducting experimentation. The code below produces a regular pattern.

```
x <- rep(seq(0, 30, by = 5), times = 7)
y <- rep(seq(0, 30, by = 5), each = 7)
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
```

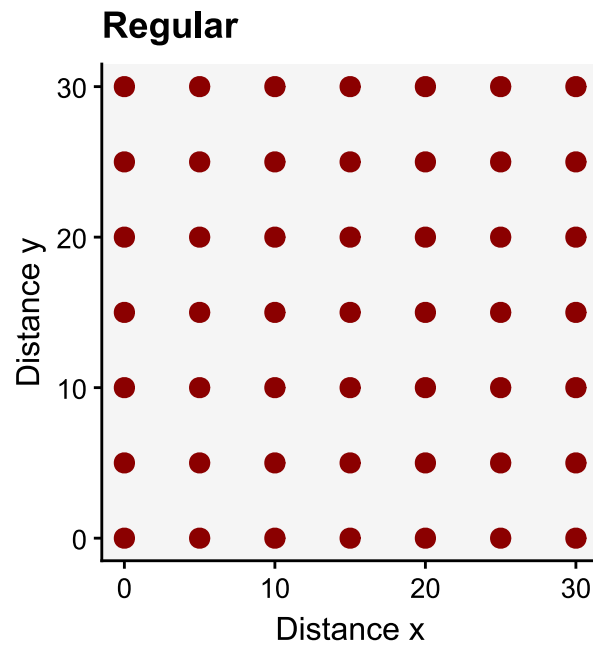


Figure 16.3: Regular pattern of a plant disease epidemic

```
library(patchwork)
preg + pr + pag
ggsave("imgs/spatial.png", width = 10, height = 4)
```

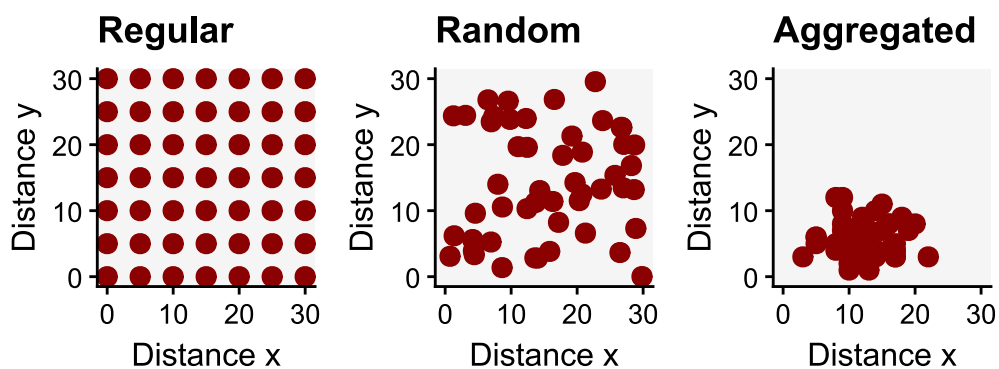


Figure 16.4: Patterns of a plant disease epidemic

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) at six times during the epidemic. The data are 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)
```

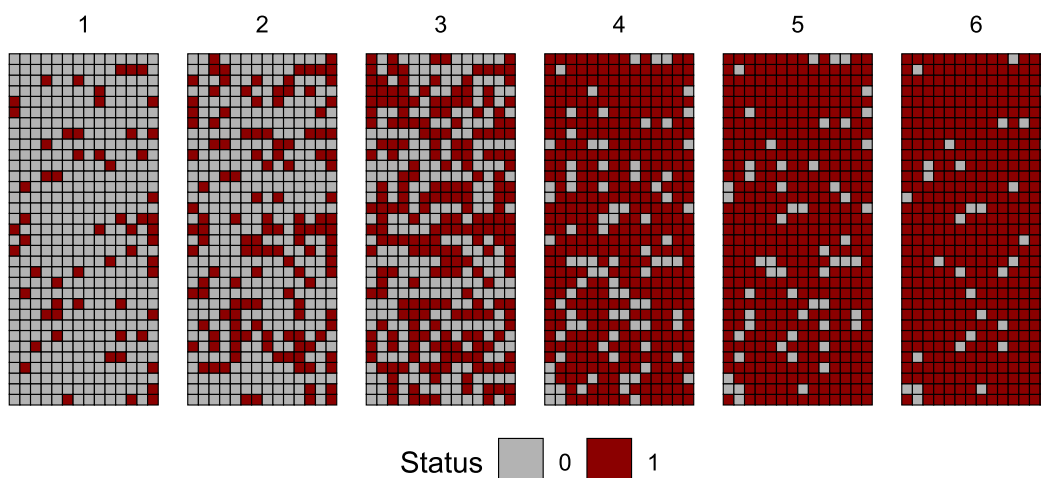


Figure 16.5: Spatial patterns of tomato spotted wilt at six assessment times

In this other example, the severity of gummy stem blight (*Didymella bryoniae*) of watermelon (Café-Filho et al., 2010) was recorded on 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 <- DidymellaWatermelon
```

```
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)
```

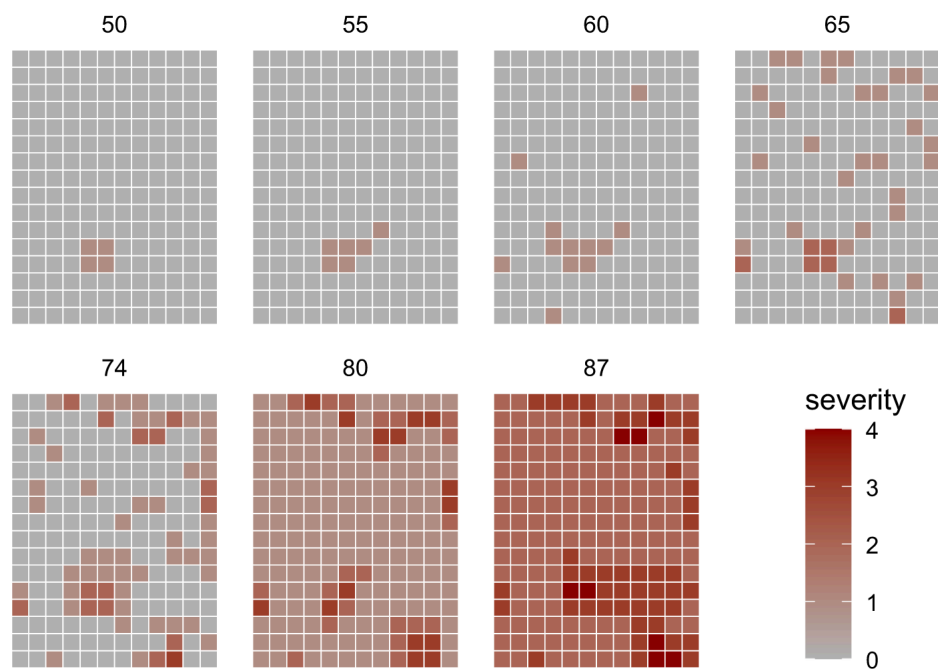


Figure 16.6: Spatial patterns of the severity (0-4 ordinal scale) of gummy stem blight of watermelon at seven times after planting (Café-Filho et al., 2010)

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.

Simulating spatial patterns

Two Shiny apps have been developed to allow simulating various spatial disease patterns. The first generates 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.

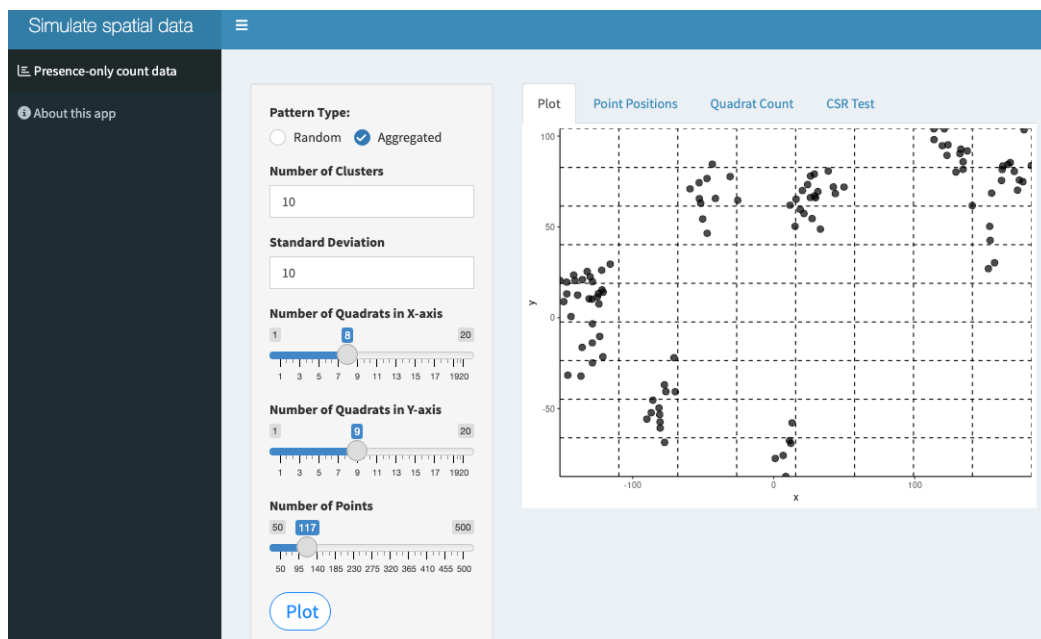


Figure 16.7: Screenshot of a Shiny app to simulate disease-only data in a grid

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).

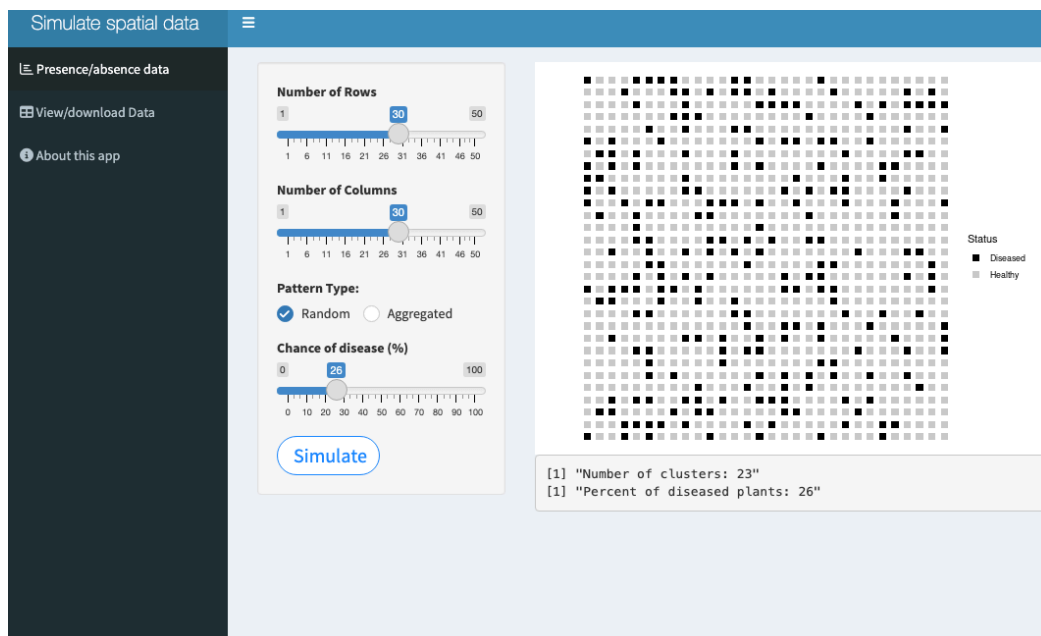


Figure 16.8: Screenshot of a Shiny app to simulate presence-absence data in a 2D map

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. \(2007a\)](#), 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.

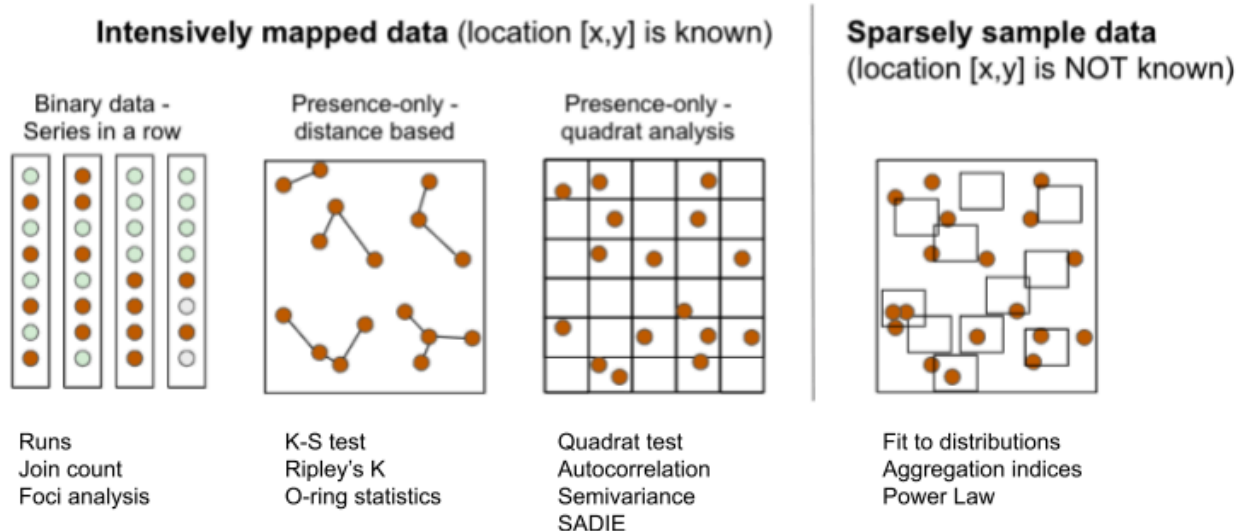


Figure 17.1: Classification of spatial data and methods used to study spatial patterns of plant disease based on knowledge of the location of the sampling units and nature of the data.

Intensively mapped

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.

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 (Madden, 1982). 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 32 plants and assign it to `y`. For plotting purposes, we make a data frame for more complete information.

```
library(tidyverse)
```

```
y1 <- c(1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,1,0,0,1,1,1,1,0,0,0)
x1 <- c(1:32) # 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 8, aided by the different colors used to identify a run.

```
library(grid)
runs2 <- row1 |>
  ggplot(aes(x1, z1, label = x1, color = factor(y1))) +
  geom_point(shape = 15, size = 10) +
  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_void() +
  theme(legend.position = "top") +
  labs(color = "Status")
ggsave("imgs/runs2.png", runs2, width = 12, height = 2, bg = "white")
```

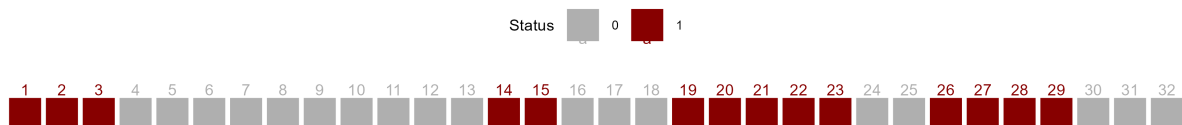


Figure 17.2: Sequence of diseased (dark red) or non-diseased (gray) units (plants). The numbers represent the position of the unit

We can obtain the number of runs and related statistics using the `oruns_test()` function of the `{r4pde}`.

```
library(r4pde)
oruns_test(row1$y1)
```

Ordinary Runs Test of Data Sequence:

```
-----
Total Number of Runs (U): 8
Expected Number of Runs (EU): 16.75
Standard Deviation of Runs (sU): 2.74
Z-score: -3.20
P-value: 0.0007
```

Interpretation:

Based on the Z-score, the sequence exhibits 'aggregation or clustering'.

Join count

In join-count statistics, two adjacent plants may be classified by the type of join that links them: D-D, H-H or H-D. 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 provide a basic measure of spatial autocorrelation. The expected number of join counts can be defined under randomness and the corresponding standard errors can be estimated after constants are calculated based on the number of rows and columns of the matrix.

Let's use the `join_count()` function of the `{r4pde}` package to perform a join count test. The formulations in this function apply only for a rectangular array with no missing values, with the "rook" definition of proximity, and they were all presented in the book *The Study of Plant Disease Epidemics*, page 261 (Madden et al., 2007c).

Let's create a series of binary data from left to right and top to bottom. The data is displayed in Fig. 9.13 in page 260 of the book (Madden et al., 2007c). In the example, there are 5 rows and 5 columns. This is specified later to compose the matrix which is the data format for analysis.

```
m1 <- 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)
matrix1 <- matrix(m1, 5, 5, byrow = TRUE)
```

We can visualize the two-dimensional array by converting to a raster.

```
# Convert to raster
mapS2 <- terra::rast(matrix1)
# 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", linewidth = 0.5) +
  theme_void() +
  coord_fixed()+
  labs(fill = "Status") +
  scale_fill_manual(values = c("gray70", "darkred"))+
  theme(legend.position = "top")
```

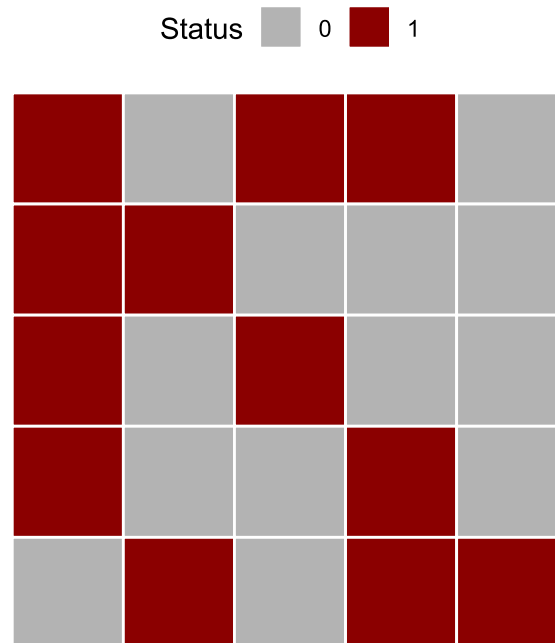



Figure 17.3: Visualization of a matrix of presence/absence data representing a disease spatial pattern

The function `join_count()` calculates spatial statistics for a matrix based on the specifications and calculations shown in the book (Madden et al., 2007c). It identifies patterns of aggregation for values in a binary matrix based on join count statistics. The results determine whether the observed spatial arrangement is aggregated or non-aggregated (random) based on a standard normal distribution test statistic Z-score applied separately for HD or DD sequences. For HD, Z-score lower than -1.64 (more negative) is taken as a basis for rejection of hypothesis of randomness ($P = 0.05$). For DD sequences, Z-score greater than 1.64 indicates aggregation.

```
library(r4pde)
join_count(matrix1)
```

Join Count Analysis of Spatial Patterns of Plant Diseases:

```
-----
1) 'HD' Sequences:
  - Observed Count: 23
  - Expected Count: 19.97
  - Standard Deviation: 3.17
  - Z-score: 0.80
  - Pattern: not aggregated

2) 'DD' Sequences:
  - Observed Count: 7
  - Expected Count: 9.22
  - Standard Deviation: 4.23
  - Z-score: -0.41
  - Pattern: not aggregated
-----
```

In the example above, the observed count of HD sequence was larger than the expected count, so the Z-score suggests non-aggregation, or randomness. The DD sequence observed count was lower than expected count, confirming non-aggregation. 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 of aggregation of diseased plants, because the observed DD is greater than expected and observed HD is lower than expected. The Z-score for HD is less than -1.64 ($P < 0.05$) and the Z-score for DD is greater than 1.64 ($P < 0.05$).

```
m2 <- 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)
matrix2 <- matrix(m2, 5, 5, byrow = TRUE)

join_count(matrix2)
```

Join Count Analysis of Spatial Patterns of Plant Diseases:

```
-----
1) 'HD' Sequences:
- Observed Count: 7
- Expected Count: 19.97
- Standard Deviation: 3.17
- Z-score: -4.24
- Pattern: aggregated

2) 'DD' Sequences:
- Observed Count: 17
- Expected Count: 9.22
- Standard Deviation: 4.23
- Z-score: 1.96
- Pattern: aggregated
-----
```

We can apply these tests for a real example epidemic data provided by the {r4pde} R package. Let's work with a portion of the intensively mapped data on the occurrence of gummy stem blight (GSB) of watermelon (Café-Filho et al., 2010).

```
library(r4pde)
gsb <- DidymellaWatermelon
gsb
```

```
# A tibble: 1,344 × 4
  dap NS_col EW_row severity
  <int> <int> <int> <int>
1    50     1     1       0
2    50     1     2       0
3    50     1     3       0
4    50     1     4       0
5    50     1     5       0
6    50     1     6       0
7    50     1     7       0
8    50     1     8       0
9    50     1     9       0
10   50     1    10       0
# i 1,334 more rows
```

The inspection of the data frame shows four variables where `dap` is the number of days after planting, `NS_col` is the north-south direction, `EW_row` is the east-west direction and `severity` is the severity in ordinal score (0-4). Let's produce a map for the 65 and 74 `dap`, but first we need to create the incidence variable based on severity.

```
gsb2 <- gsb |>
  filter(dap %in% c(65, 74)) |>
  mutate(incidence = case_when(severity > 0 ~ 1,
                                TRUE ~ 0))

gsb2 |>
  ggplot(aes(NS_col, EW_row, fill = factor(incidence)))+
  geom_tile(color = "white") +
  coord_fixed() +
  scale_fill_manual(values = c("gray70", "darkred")) +
  facet_wrap( ~ dap) +
  labs(fill = "Status")+
  theme_void()+
  theme(legend.position = "top")
```

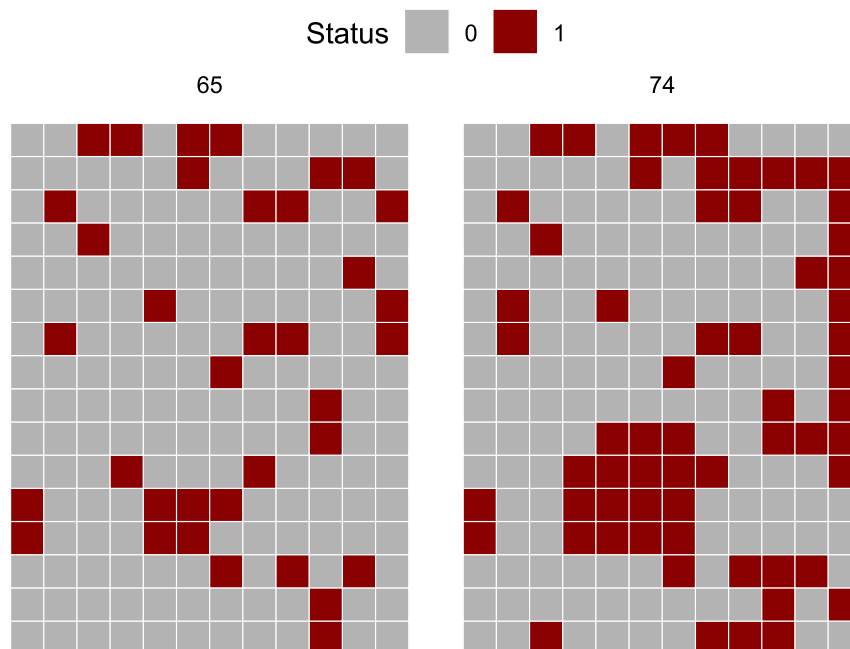


Figure 17.4: Incidence maps for gummy stem blight of watermelon

Now we should check the number of rows (y) and columns (x) for further preparing the matrix for the join count statistics. The `m65` object should be in the matrix format before running the join count test.

```
max(gsb2$NS_col)
```

```
[1] 12
```

```
max(gsb2$EW_row)
```

```
[1] 16
```

```
m65 <- gsb2 |>
  filter(dap == 65) |>
  pull(incidence) |>
  matrix(12, 16, byrow = TRUE)

join_count(m65)
```

```
Join Count Analysis of Spatial Patterns of Plant Diseases:
```

```
-----
1) 'HD' Sequences:
- Observed Count: 104
- Expected Count: 108.47
- Standard Deviation: 13.89
- Z-score: -0.36
- Pattern: not aggregated

2) 'DD' Sequences:
- Observed Count: 15
- Expected Count: 12.52
- Standard Deviation: 4.76
- Z-score: 0.63
- Pattern: not aggregated
-----
```

We can apply the join count test for 74 dap. The result shows that the pattern is now aggregated, differing from 65 dap.

```
# Pull the binary sequence of time 2
m74 <- gsb2 |>
  filter(dap == 74) |>
  pull(incidence) |>
  matrix(12, 16, byrow = TRUE)

join_count(m74)
```

```
Join Count Analysis of Spatial Patterns of Plant Diseases:
```

```
-----
1) 'HD' Sequences:
- Observed Count: 110
- Expected Count: 155.67
- Standard Deviation: 11.91
- Z-score: -3.88
- Pattern: aggregated

2) 'DD' Sequences:
- Observed Count: 58
- Expected Count: 37.12
- Standard Deviation: 8.85
- Z-score: 2.42
- Pattern: aggregated
-----
```

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 non-diseased trees are mapped in the 2D plane (Jesus Junior & 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 × 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")

```

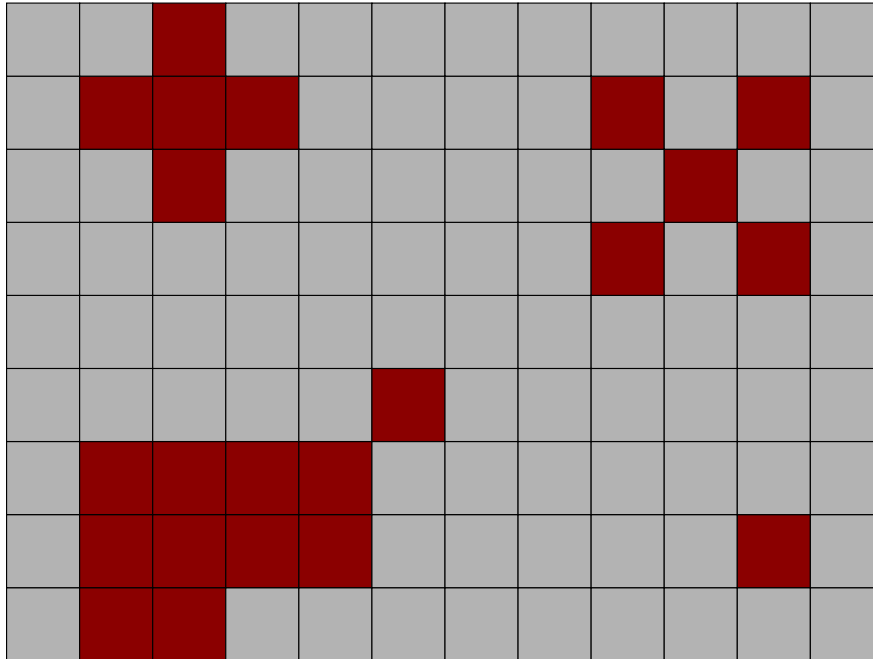


Figure 17.5: Examples of foci of plant diseases - see text for description

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 diagonal proximity to the central unit (or the Bishop's case). The lower left focus is composed of 11 diseased plants with 5 rows and 4 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-plant foci (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 (NPF $_i$)
- Maximum number of rows of the focus i (rfi) and maximum number of columns of the focus i (cfi)
- Mean shape index of foci ($\text{meanSIF} = [\sum(\text{rfi} / \text{cfi})] / \text{NF}$), 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 ($\text{meanCIF} = [\sum(\text{NPFi}/\text{rfi} \cdot \text{cfi})]/\text{NF}$), where CIF values close to 1.0 indicate 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 × 3] (S3: tbl_df/tbl/data.frame)
 $ x: num [1:108] 1 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 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 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
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

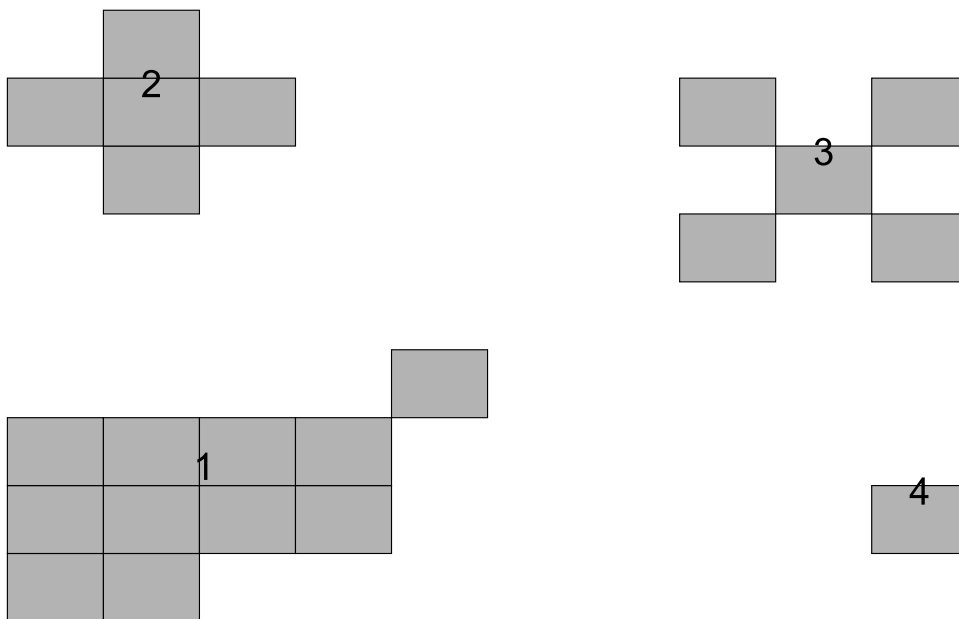
The third object is the original data frame amended with the id for each focus which can be plotted and labelled (the focus ID) using the `plot_AFSD()` function.

```
foci_data <- result_foci[[3]]
foci_data
```

```
# A tibble: 22 × 4
      x     y     i focus_id
  <dbl> <int> <dbl>   <dbl>
1     2     1     1       1
2     2     2     1       1
3     2     3     1       1
4     2     8     1       2
5     3     1     1       1
6     3     2     1       1
7     3     3     1       1
8     3     7     1       2
9     3     8     1       2
10    3     9     1       2
# i 12 more rows
```

The plot shows the ID for each focus.

```
plot_AFSD(foci_data)+
  theme_void()
```



We will now analyze the gummy stem blight dataset again focusing on 65 and 74 dap. We first need to prepare the data for the analysis by getting the x, y and i vectors in the dataframe.

```
library(r4pde)
gsb65 <- DidymellaWatermelon |>
  filter(dap %in% c(65)) |>
  mutate(incidence = case_when(severity > 0 ~ 1,
                                TRUE ~ 0)) |>
  select(EW_row, NS_col, incidence) |>
  rename(x = EW_row, y = NS_col, i = incidence)
```

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


```
result_df1 <- AFSD(gsb65)

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

stats	value
NF	12.0000000
NF1000	62.5000000
NSF	2.0000000
NSF1000	10.4166667
DIS_INC	0.1875000
mean_SIF	1.0722222
mean_CIF	0.7162037

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 the two dap. We can split it by dap before applying the function. We can do it using the `map` function of the *purrr* package.

```
library(purrr)
gsb65_74 <- DidymellaWatermelon |>
  filter(dap %in% c(65, 74)) |>
  mutate(incidence = case_when(severity > 0 ~ 1,
                                TRUE ~ 0)) |>
  select(dap, EW_row, NS_col, incidence) |>
  rename(dap = dap, x = EW_row, y = NS_col, i = incidence)

# Split the dataframe by 'time'
df_split <- split(gsb65_74, gsb65_74$dap)

# 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.

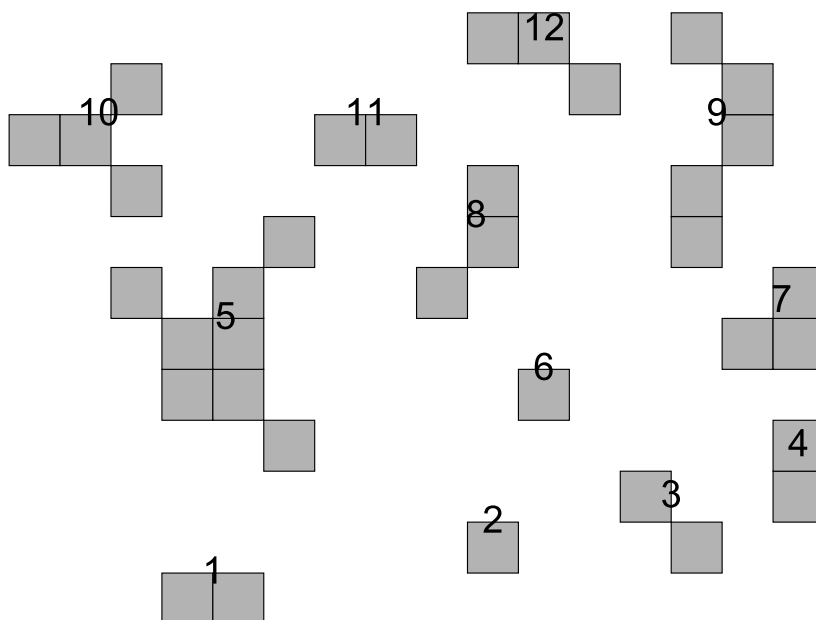
```
time65 <- data.frame(results[[1]][1])
knitr::kable(time65)
```

stats	value
NF	12.0000000
NF1000	62.5000000
NSF	2.0000000
NSF1000	10.4166667
DIS_INC	0.1875000
mean_SIF	1.0722222
mean_CIF	0.7162037

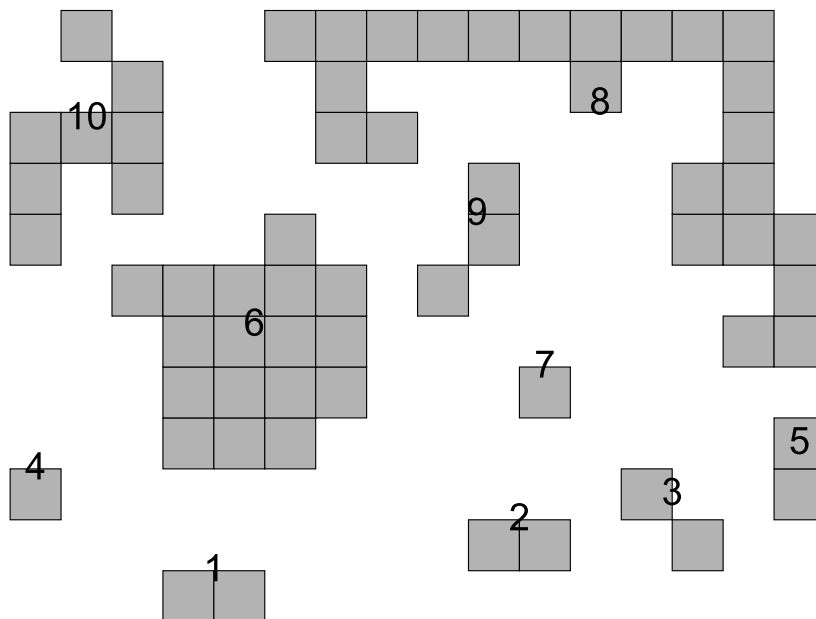
```
time74 <- data.frame(results[[2]][1])
knitr::kable(time74)
```

stats	value
NF	10.0000000
NF1000	52.0833333
NSF	2.0000000
NSF1000	10.4166667
DIS_INC	0.3229167
mean_SIF	1.1338095
mean_CIF	0.7525022

```
# 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()
```



Point pattern analysis

Point pattern analysis studies the arrangement of points in a two-dimensional space. In the simplest case each point is the location of a diseased plant, and the pattern is a map of these locations. A map (a scatter plot with equally scaled axes) gives a first impression, but we need quantitative methods to decide whether the points are random, clustered (closer together than expected by chance) or regular (more evenly spaced than expected by chance). Each of these patterns points to different processes behind the epidemic.

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

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

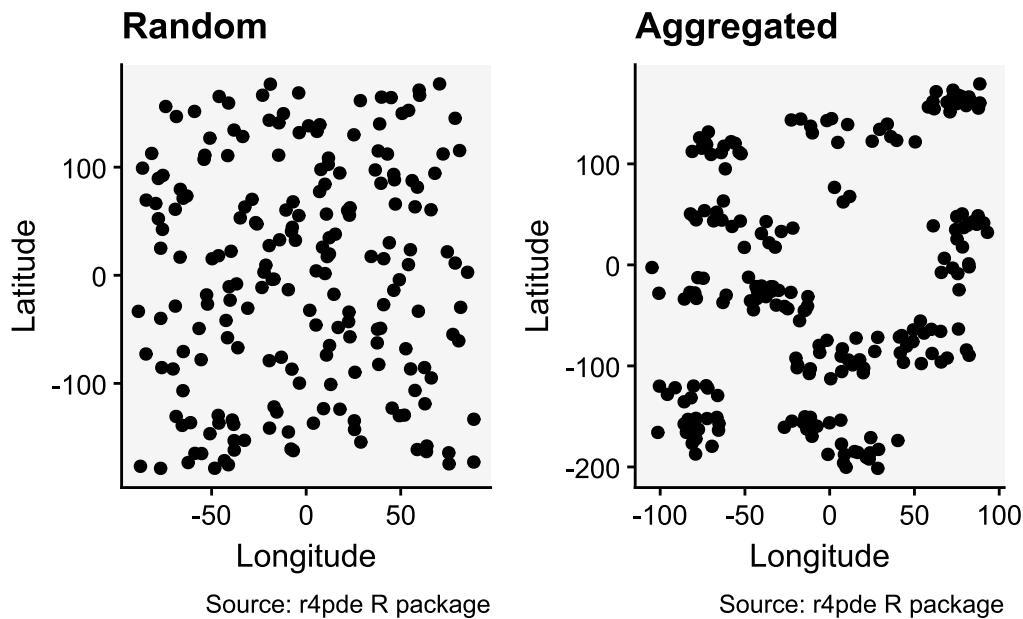
Let's produce a 2-D map for each data frame.

```
prand <- rand |>
  ggplot(aes(x, y))+
  geom_point()+
  scale_color_manual(values = c("black", NA))+
  coord_fixed()+
  coord_flip()+
  theme_r4pde(font_size = 12)+
  theme(legend.position = "none")+
  labs (title = "Random",
        x = "Latitude",
        y = "Longitude",
        caption = "Source: r4pde R package")

paggr <- aggr |>
  ggplot(aes(x, y))+
  geom_point()+
  scale_color_manual(values = c("black", NA))+
```

```
coord_fixed()+
coord_flip()+
theme_r4pde(font_size = 12)+
theme(legend.position = "none")+
labs (title = "Aggregated",
      x = "Latitude",
      y = "Longitude",
      caption = "Source: r4pde R package")

library(patchwork)
prand | paggr
```



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.

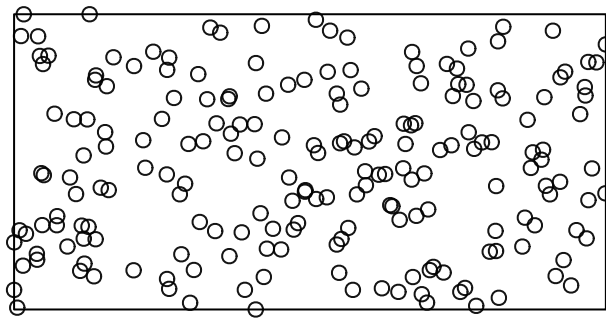
Using the `{spatstat}` package, we first need to create a `ppp` object which represents a point pattern. This is the primary object type in `spatstat` for point patterns.

```
library(spatstat)

### Create rectangular window around the points
window_rand <- bounding.box.xy(rand$x, rand$y)

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

ppp_rand



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

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

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

qq

3	1	2	1	3	0	3	2
3	6	3	3	2	5	2	5
3	1	5	3	3	3	4	3
1	3	4	2	7	4	4	3
4	3	3	3	5	4	2	5
6	1	2	6	2	5	2	3
8	2	3	2	2	3	3	2
2	1	3	3	3	4	4	2

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 = 47.04, p-value = 0.937
alternative hypothesis: clustered

Quadrats: 8 by 8 grid of tiles
```

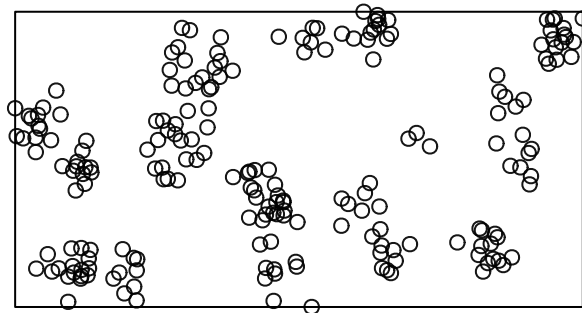
The result will give a Chi-squared statistic and a p-value. If the p-value is very low, then the pattern is likely not random. Keep in mind that the choice of the number and size of quadrats can affect the results. It's often helpful to try a few different configurations to ensure robust conclusions.

Let's repeat this procedure for the situation of aggregated data.

```
### Create a rectangular window around the points
window_aggr <- bounding.box.xy(aggr$x, aggr$y)

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

ppp_aggr



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

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

qq_aggr

0	0	4	1	8	9	0	13
0	0	9	1	2	1	1	4
7	1	7	0	0	0	3	2
8	2	9	0	0	3	1	3
8	3	5	9	2	0	1	3
0	0	0	13	4	2	2	0
7	6	0	6	0	7	10	1
6	7	0	3	1	2	1	0

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

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

```
data:
X2 = 244.92, p-value = 5e-04
alternative hypothesis: clustered
```

```
Quadrats: 8 by 8 grid of tiles
```

Spatial KS test

The spatial Kolmogorov-Smirnov (KS) test compares the distribution of a spatial covariate at the observed points with the distribution expected if the points were completely spatially random (CSR) (Baddeley et al., 2005). If the points tend to lie where the covariate takes particular values (for example, low elevation or close to a field edge), the test rejects CSR.

The covariate can be any spatially varying quantity that might influence the process, such as elevation, soil quality or distance to a feature. Here we use the coordinates themselves as covariates, so the test asks whether the distribution of the x-coordinates, the y-coordinates, or a combination of both, differs from what is expected under CSR.

Let's test for the aggregated data.

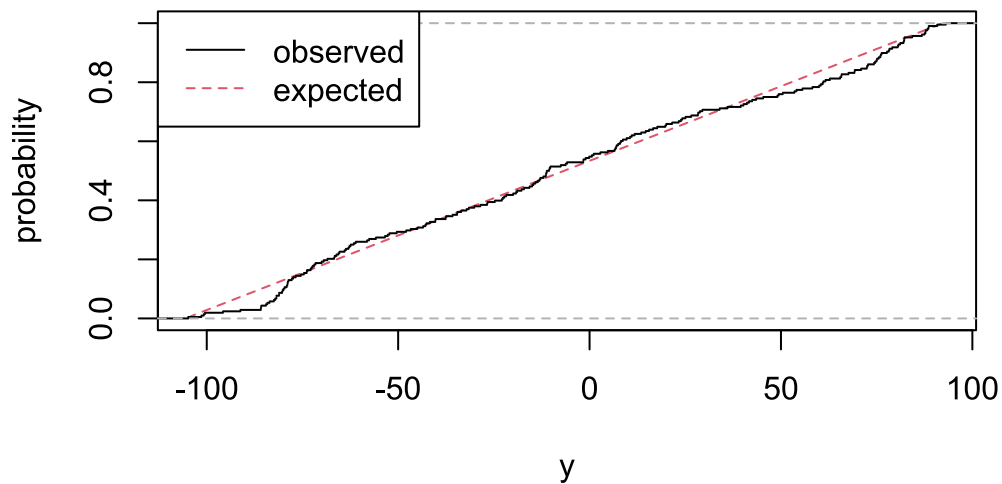
```
# y as covariate
ks_y <- cdf.test(ppp_aggr, test="ks", "y", jitter=FALSE)
ks_y
```

```
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 = 0.070995, p-value = 0.2453
alternative hypothesis: two-sided
```

```
plot(ks_y)
```

Spatial Kolmogorov-Smirnov test of CSR in two dimensions based on distribution of y coordinate p-value= 0.2453



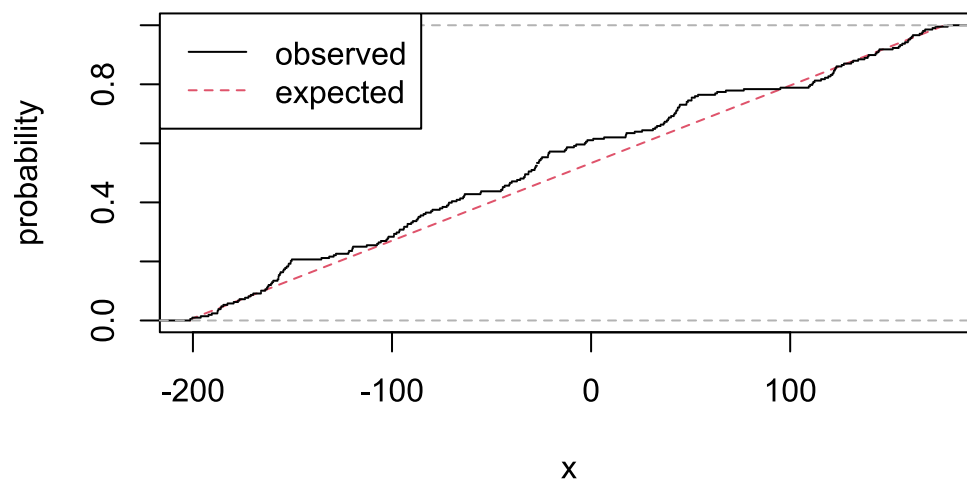
```
# x as covariate
ks_x <- cdf.test(ppp_aggr, test="ks", "x", jitter=FALSE)
ks_x
```

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 = 0.093863, p-value = 0.0512
alternative hypothesis: two-sided
```

```
plot(ks_x)
```


Spatial Kolmogorov-Smirnov test of CSR in two dimensions based on distribution of x coordinate p-value= 0.0512



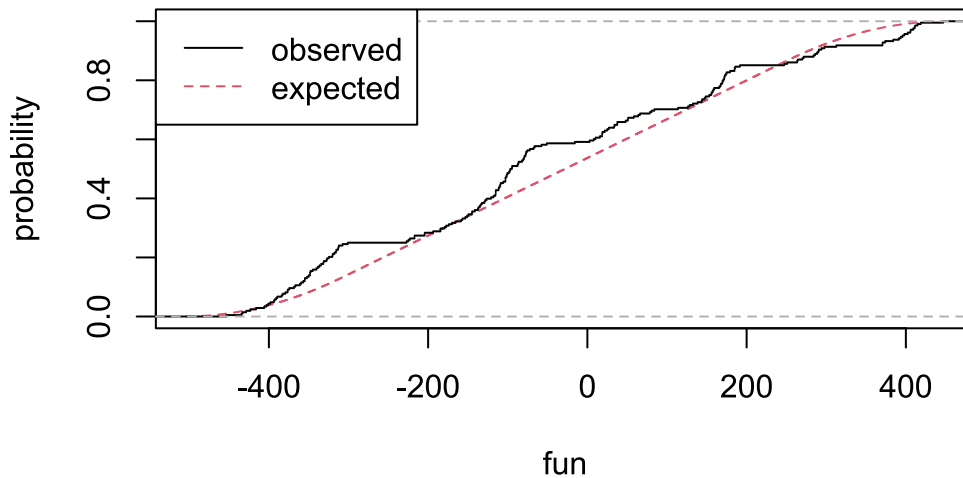
```
# x and y as covariates
fun <- function(x,y){2* x + y}
ks_xy <- cdf.test(ppp_aggr, test="ks", fun, jitter=FALSE)
ks_xy
```

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 = 0.12687, p-value = 0.002471
alternative hypothesis: two-sided
```

```
plot(ks_xy)
```

Spatial Kolmogorov-Smirnov test of CSR in two dimensions based on distribution of covariate "fun" p-value= 0.002471



The tests based on the x- or y-coordinate alone do not clearly reject complete spatial randomness at the 5% level ($P = 0.051$ and $P = 0.25$), but the test based on the combined covariate does ($P = 0.002$). We therefore have evidence against the null hypothesis of complete spatial randomness.

Distance based

Ripley's K

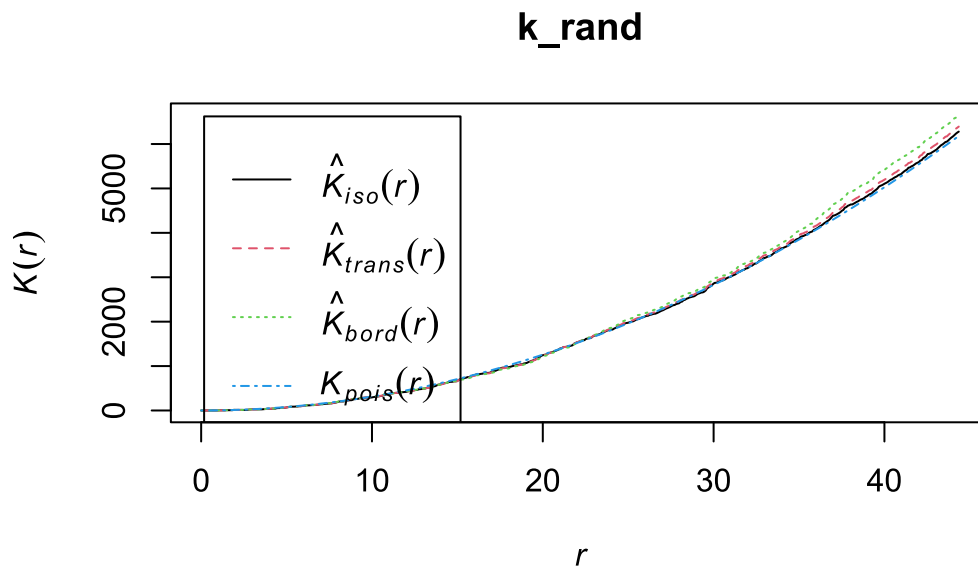
A spatial point process is a set of irregularly distributed locations in a region, assumed to be generated by a stochastic mechanism. Ripley's K-function, $K(r)$, describes the process at all distances at once. Assuming a constant intensity of points, $K(r)$ is the expected number of further points within distance r of an arbitrary point, divided by the intensity. For diseased trees in a plantation, it answers the question: how many other diseased trees do we expect within distance r of a diseased tree chosen at random?

To interpret the estimated function:

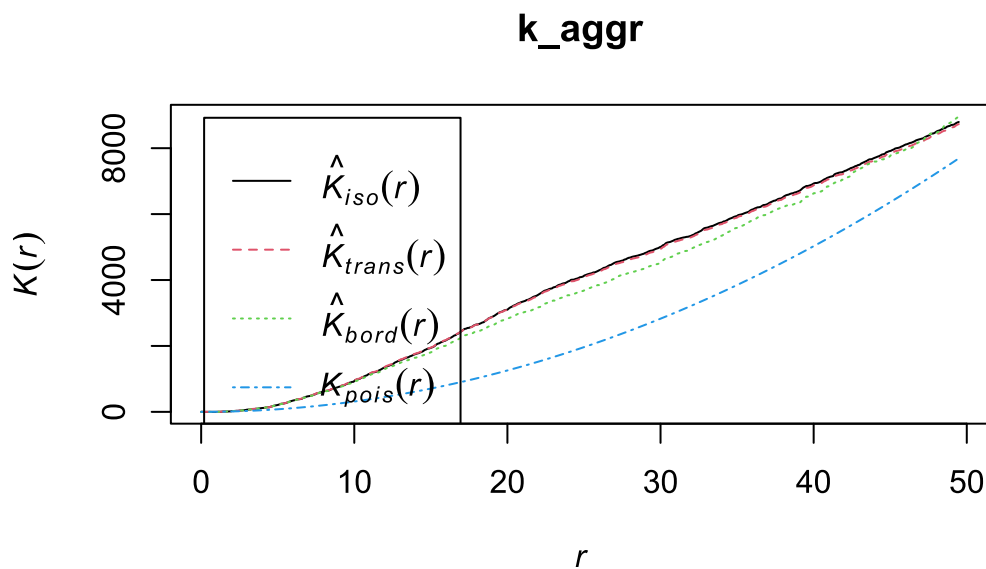
1. **Random pattern:** $K(r)$ follows the theoretical curve for complete spatial randomness, $K(r) = \pi r^2$.
2. **Clustered pattern:** $K(r)$ lies above the theoretical curve, because points have more neighbors than expected by chance.
3. **Regular pattern:** $K(r)$ lies below the theoretical curve, because points have fewer neighbors than expected by chance.

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) lies outside the simulation envelope, aggregation was detected.

```
ke <- envelope(ppp_aggr, fun = Kest)
```

```
Generating 99 simulated realisations 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, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
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)
```

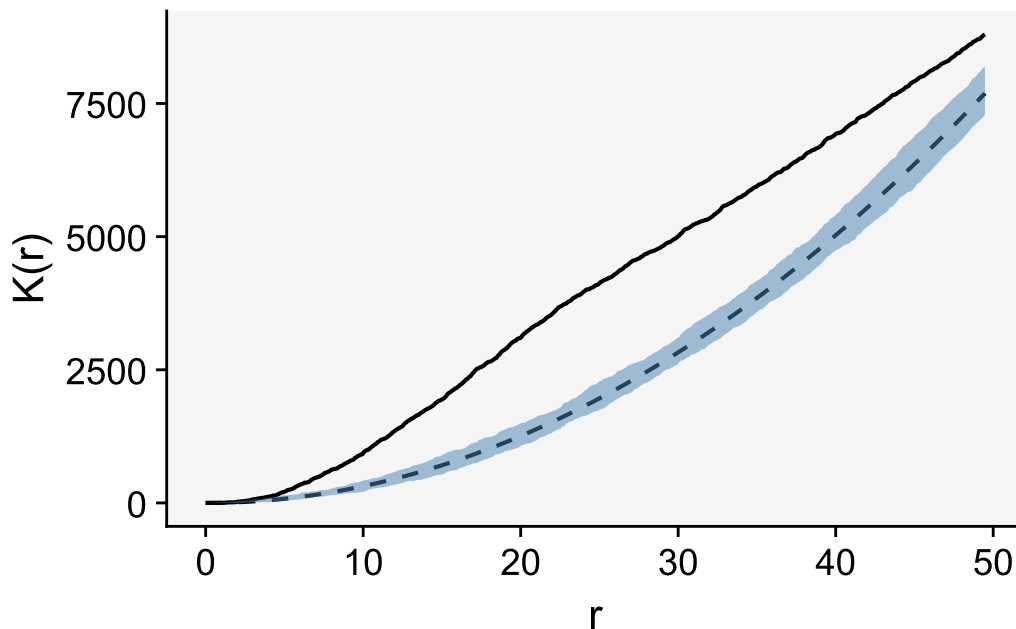


Figure 17.6: Estimated $K(r)$ of the aggregated pattern (solid line) and the simulation envelope under complete spatial randomness (shaded area); the dashed line is the theoretical $K(r)$

`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 simulated realisations 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, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
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, 49.477517555]
Test statistic: Maximum absolute deviation
Deviation = observed minus theoretical
```

```
data: ppp_aggr
mad = 2234.7, rank = 1, p-value = 0.01
```

```
mad.test(ppp_rand, Kest)
```

```
Generating 99 simulated realisations 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, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
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, 44.352774865]
Test statistic: Maximum absolute deviation
Deviation = observed minus theoretical
```

```
data: ppp_rand
mad = 106.7, rank = 92, p-value = 0.92
```

O-ring statistics

The O-ring statistic estimates the average density of points in a ring of radius r around the points of a pattern, so it shows at which distances points are more or less frequent than expected (Wiegand & A. Moloney, 2004). In the univariate form used here the rings are placed around every point of a single pattern. In the bivariate form the rings are placed around points of one type and the points of a second type are counted.

The statistic is interpreted relative to the intensity of the pattern, which is the value expected under CSR:

1. **Random pattern:** $O(r)$ is flat and close to the intensity at all distances.
2. **Aggregation:** $O(r)$ is above the intensity at short distances and decreases toward it as r increases, because points are surrounded by more neighbors than expected.
3. **Regularity:** $O(r)$ is below the intensity at short distances and increases toward it as r increases.

As with $K(r)$, a simulation envelope under CSR is used for inference: if the observed statistic lies inside the envelope, the pattern is consistent with randomness, and if it lies outside, it is not.

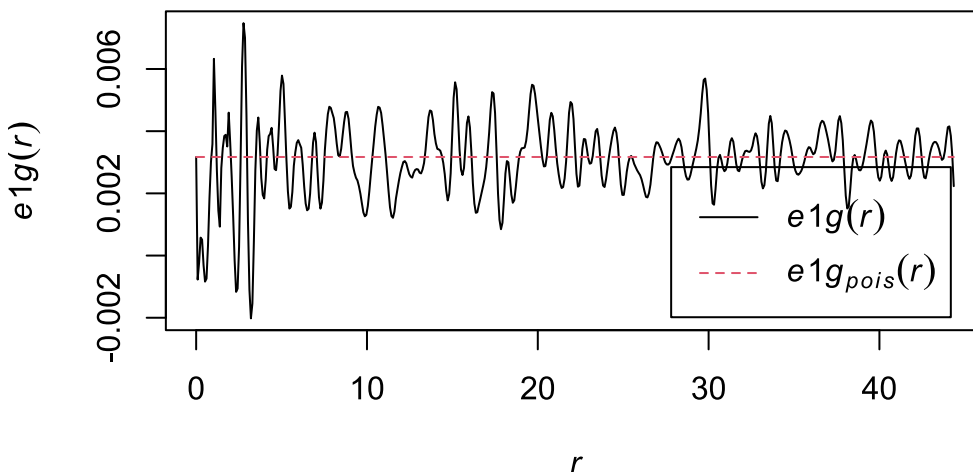
In R, the O-ring statistic is $O(r) = \lambda g(r)$, the intensity λ of the pattern multiplied by the pair correlation function $g(r)$. The *onpoint* package provides a fast estimate of $g(r)$ with `pcf_fast()` (older versions of the package had a function called `estimate_o_ring()`, which we define here as a small wrapper). We will use the point pattern objects `ppp_rand` and `ppp_aggr` created in the previous examples

```
library(onpoint)

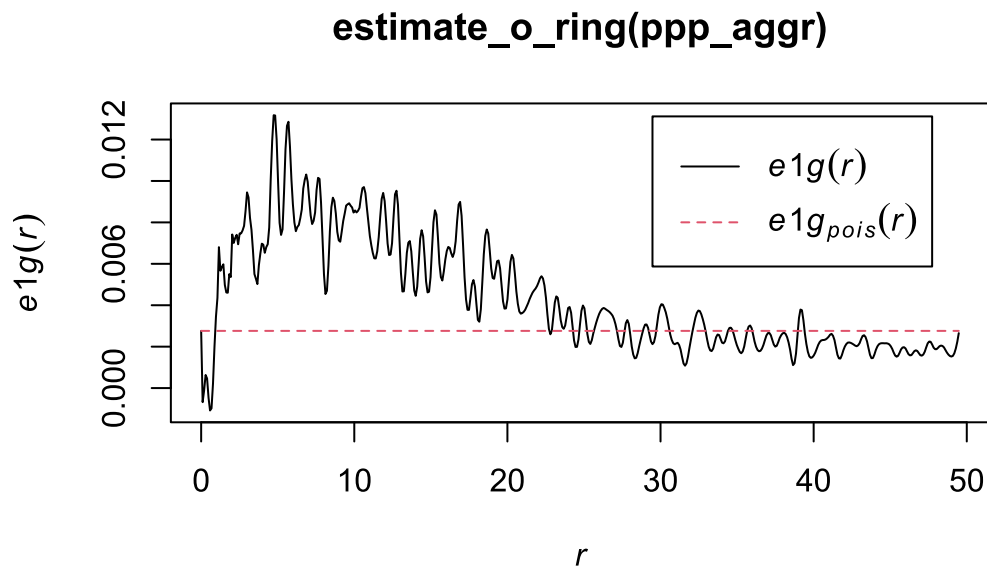
# O-ring statistic: intensity x pair correlation function
estimate_o_ring <- function(X, ...) {
  intensity(X) * pcf_fast(X, ...)
}

plot(estimate_o_ring(ppp_rand))
```

estimate_o_ring(ppp_rand)

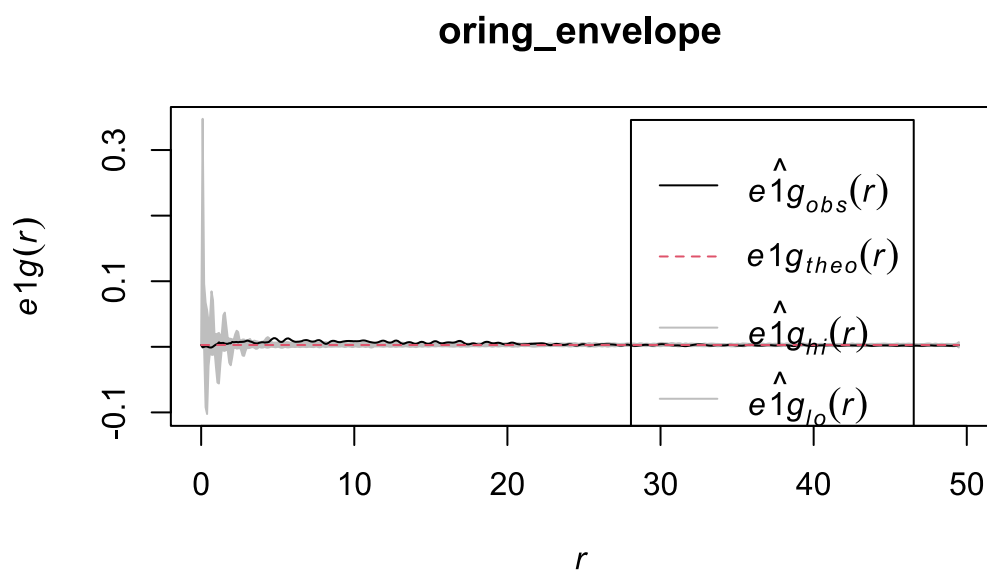


```
plot(estimate_o_ring(ppp_aggr))
```



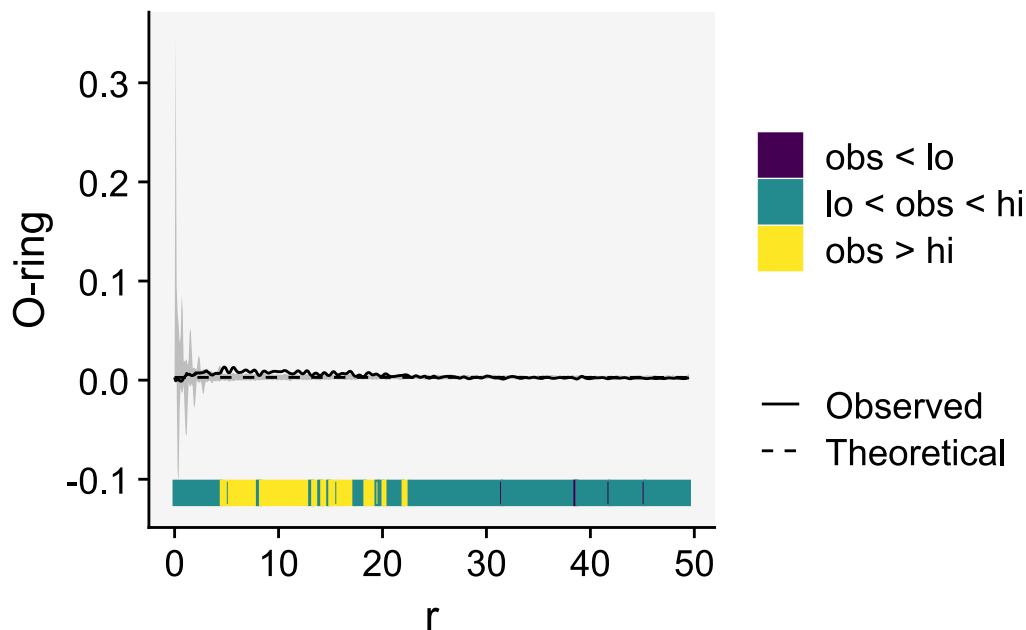
The function can be used in combination with `spatstat`'s `envelope()` function.

```
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()
```



Grouped data

If the data are intensively mapped, meaning that the spatial locations of the sampling units are known, we are not limited to analyze 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.)

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 autocorrelation) likely that neighboring sampling units tend to have a large value of disease intensity (Madden et al., 2007c).

We will illustrate the basic concept by reproducing the example provided in page 264 of the chapter on spatial analysis (Madden et al., 2007c), which was extracted from table 11.3 of Campbell & Madden. L. (1990). The data represent a single transect with the number of *Macrophomina 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

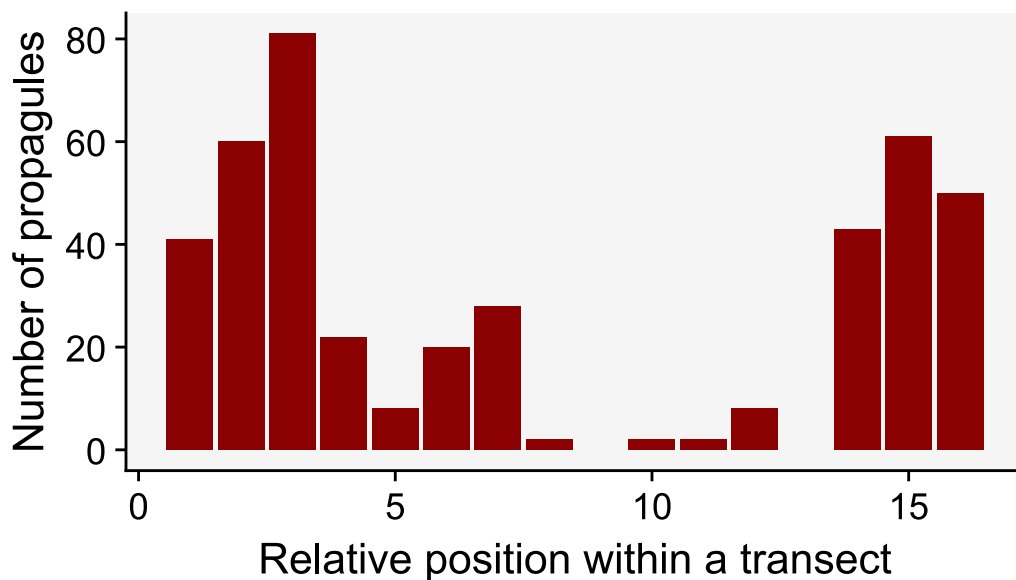
```

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)"
  )

```



Source: Campbell and Madden (1990)

Figure 17.7: 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 `acf()` function of the *stats* package (part of base R).

```
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")
```

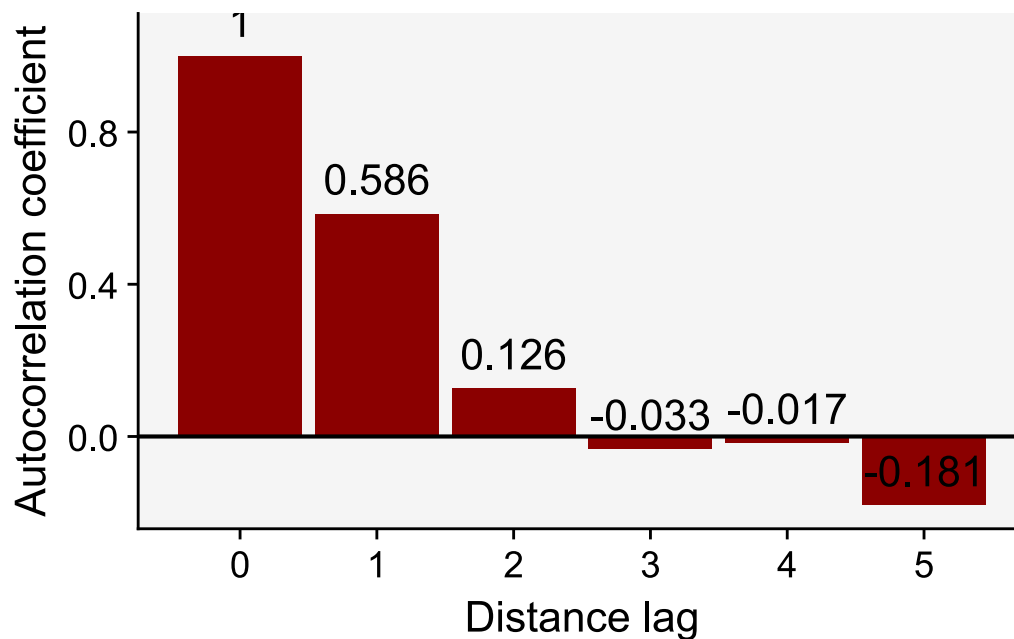


Figure 17.8: Autocorrelogram for the spatial distribution of *Macrophomina phaseolina* in soil

The values we obtained here are not the same but quite close to the values reported in [Madden et al. \(2007a\)](#). 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.

Moran's I

The autocorrelation coefficient above is closely related to Moran's I ([Moran, 1950](#)), a widely used statistic of spatial autocorrelation. Positive autocorrelation means that nearby locations have more similar values than expected under a random arrangement, and negative autocorrelation means that they have more dissimilar values.

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 & Madden \(2004\)](#). The data is stored in a CSV file.

```
epi <- read_csv("https://raw.githubusercontent.com/emdelponte/epidemiology-R/main/data/xu-madden-simulated.csv")
epi1 <- epi |>
  pivot_longer(2:13,
    names_to = "y",
    values_to = "n") |>
  pull(n)
```

The `{spdep}` package provides the function `moran()` to calculate Moran's I. It needs two inputs: a numeric vector with the values (here, the number of diseased plants in each quadrat) and a spatial weights list that defines which quadrats are neighbors. Let's load the library and create the weights.

```
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 = length(epi1), # number of zones
  S0 = Szero(col.W)) # global sum of weights
```

```
$I
[1] 0.6982314
```

```
$K
[1] 2.878088
```

`$I` is Moran's I and `$K` is the sample kurtosis of the values.

A positive Moran's I indicates positive spatial autocorrelation (nearby quadrats have similar values), a negative value indicates negative autocorrelation (neighboring quadrats tend to be dissimilar), and a value close to zero indicates no spatial autocorrelation.

Moran's test

To test whether the autocorrelation is statistically significant, we use `moran.test()` of the `{spdep}` package. Its two main inputs are `x`, the numeric vector of values, and `listw`, the spatial weights list created above.

```
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 returns Moran's I , its expected value under the null hypothesis of no spatial autocorrelation, its variance and a P-value. A significant positive value indicates that similar values cluster together, a significant negative value indicates that dissimilar values are neighbors, and a non-significant value gives no evidence of spatial autocorrelation.

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., 2007c). Let's store the results in a data frame.

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

```
Warning in nblag(neighbours, maxlag = order): lag 7 neighbour object has 5
sub-graphs
```

```
Warning in nblag(neighbours, maxlag = order): lag 8 neighbour object has 17
sub-graphs
```

```
Warning in nblag(neighbours, maxlag = order): lag 9 neighbour object has 37
sub-graphs
```

```
Warning in nblag(neighbours, maxlag = order): lag 10 neighbour object has 65
sub-graphs
```

```
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")
```

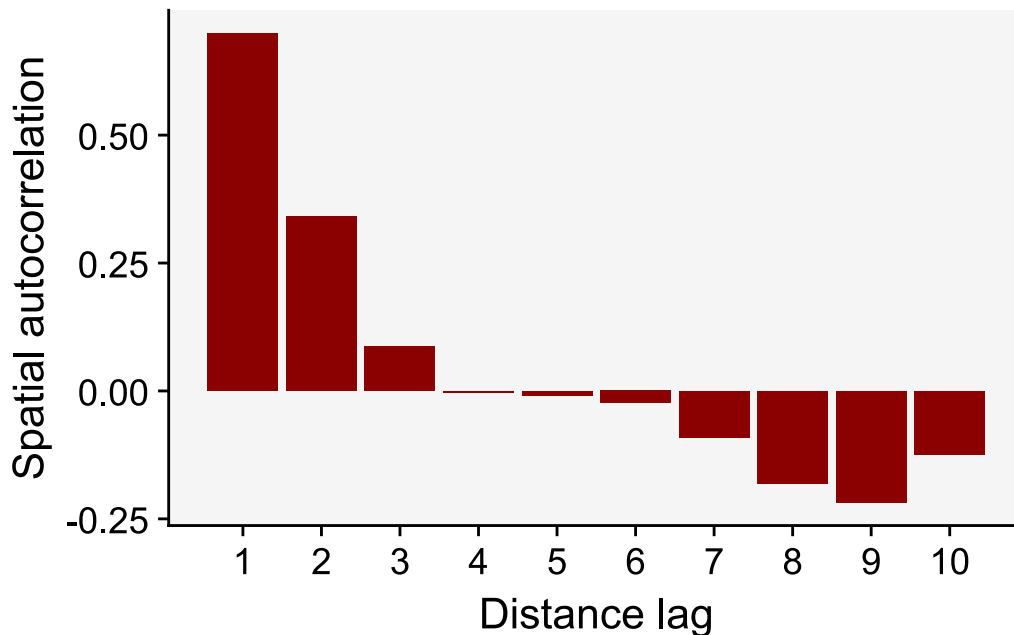


Figure 17.9: Autocorrelogram for the spatial distribution of simulated epidemics

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 × 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)
```

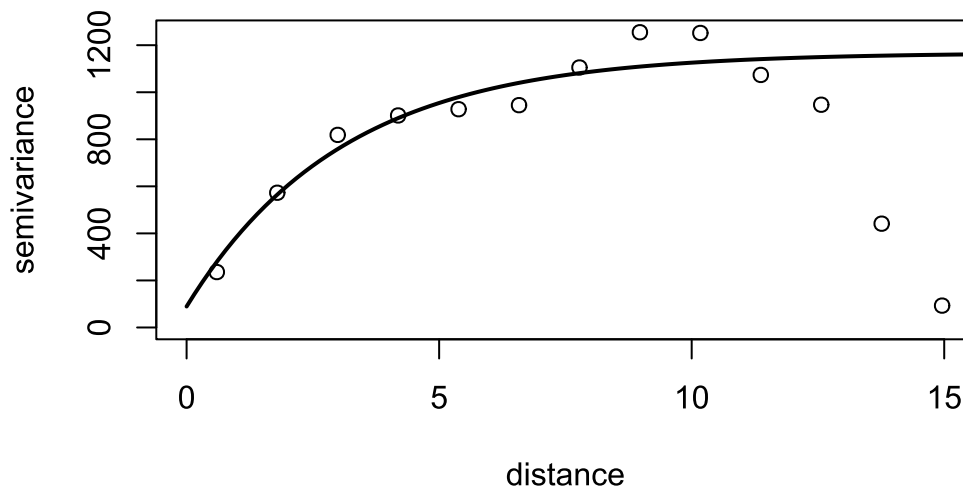


Figure 17.10: Semivariance plot for the spatial distribution simulated epidemic

SADIE

SADIE (spatial analysis by distance indices) is an alternative to autocorrelation and semi-variance methods described previously, which has found use in plant pathology (Li et al., 2011; Madden et al., 2007c; Xu & Madden, 2004). 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")

```

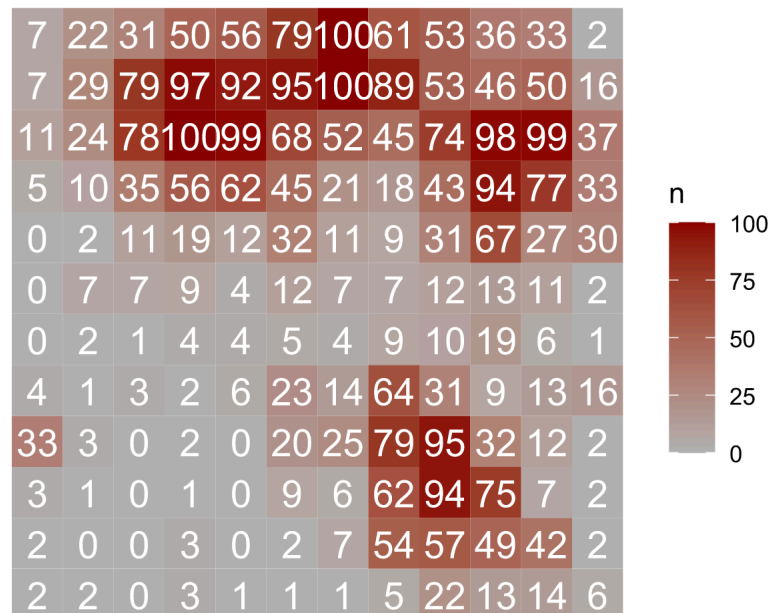


Figure 17.11: Spatial map for the number of diseased plants per quadrat ($n = 144$) in simulated epidemic

```
library(epiphy)
```

```
Attaching package: 'epiphy'
```

```
The following object is masked from 'package:dplyr':
```

```
count
```

```
The following object is masked from 'package:stats':
```

```
chisq.test
```

```
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 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 us to map the clustering indices and so to identify regions of patches (red, outflow) and gaps (blue, inflow).

```
plot(sadie_epi2)
```

```
Warning: `aes()` was deprecated in ggplot2 3.0.0.
```

```
i Please use tidy evaluation idioms with `aes()`
```

```
i The deprecated feature was likely used in the epihy package.
```

```
Please report the issue at <https://github.com/chgigot/epiphy/issues>.
```

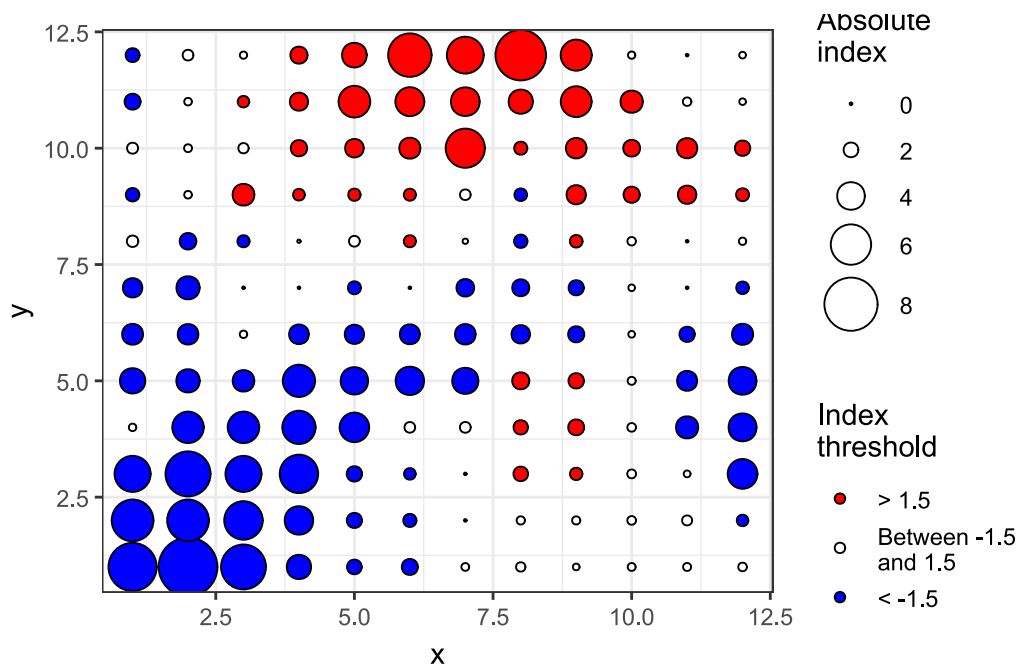


Figure 17.12: Map of the SADIE clustering indices where red identify patches (outflow) and blue identify gaps (inflow)

An isocline plot can be obtained by setting the `isoclines` argument as `TRUE`.

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

Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
 i Please use `linewidth` instead.
 i The deprecated feature was likely used in the epiphy package.
 Please report the issue at <<https://github.com/chgigot/epiphy/issues>>.

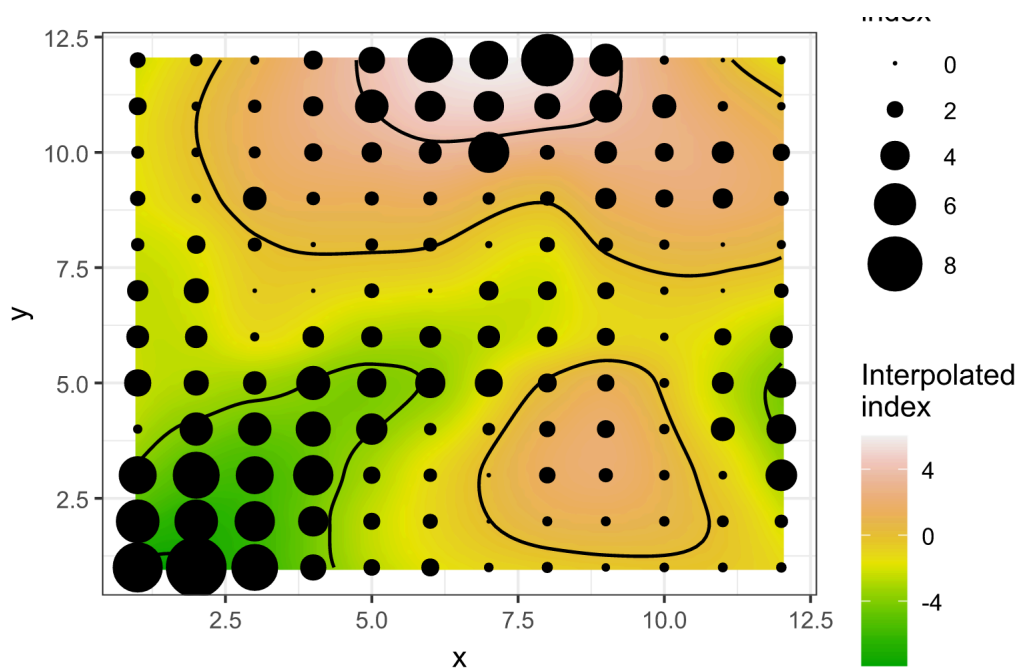


Figure 17.13: Map of the SADIE clustering indices

Sparsely sampled data

In sparsely sampled data, the locations of the sampling units are unknown or are ignored in the analysis (Madden et al., 2007c). The aim is to describe the variability of disease intensity among sampling units, and there are two approaches: (1) testing the goodness of fit of the data to a statistical distribution, and (2) calculating indices of aggregation. The distributions and indices depend on whether the data are counts or incidences (proportions), so we treat the two cases separately.

Count data

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 distributions 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 for 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   2.9510195   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 suggests 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)
```

```
Warning in geom_bar(stat = "identity", position = "dodge", color = "black", :
Ignoring unknown parameters: `size`
```

```
Warning: The `size` argument of `element_rect()` is deprecated as of ggplot2 3.4.0.
i Please use the `linewidth` argument instead.
i The deprecated feature was likely used in the epiphy package.
Please report the issue at <https://github.com/chgigot/epiphy/issues>.
```

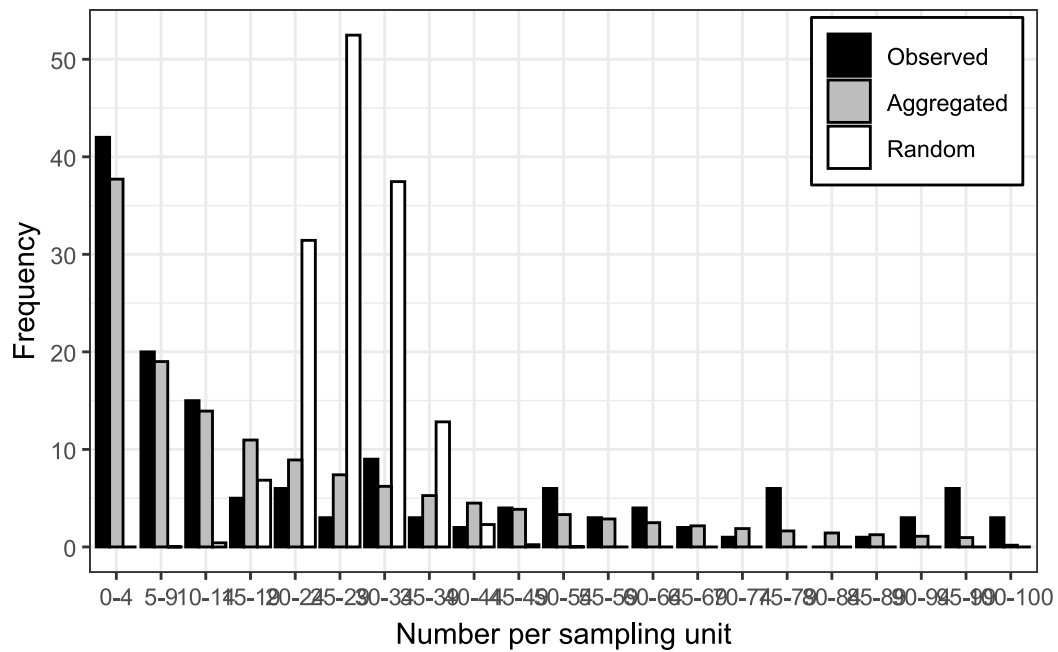


Figure 17.14: Frequencies of the observed and expected aggregated and random distributions

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")
```

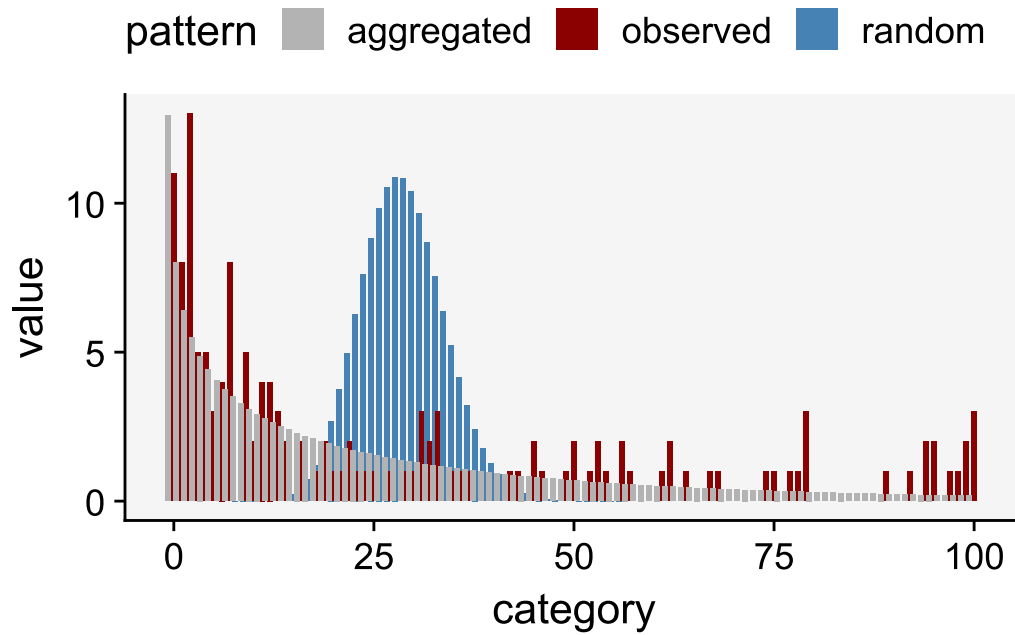


Figure 17.15: Frequencies of the observed and expected aggregated and random distributions

Aggregation indices

Several indices summarize the degree of aggregation of counts (Fisher, Lloyd and Morisita).

```
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
```

Incidence data

Fit to distributions

```
tas <-
  read.csv(
    "data/tasmania_test_1.txt",
    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

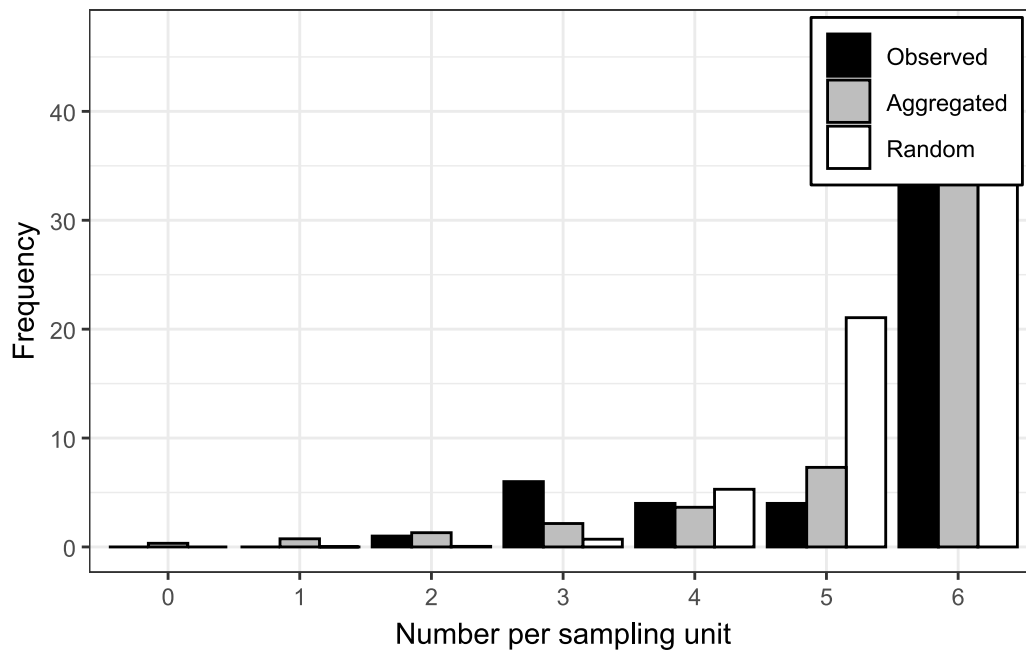
(2) Beta-binomial (aggregated):
      Estimate Std.Err Z value  Pr(>z)
alpha 1.923479 0.869621  2.2119 0.026976 *
beta   0.181337 0.075641  2.3973 0.016514 *
prob   0.913847 0.023139 39.4943 < 2.2e-16 ***
rho    0.322080 0.096414  3.3406 0.000836 ***
theta  0.475101 0.209789  2.2647 0.023534 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
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
```

```
plot(fit_tas)
```

Aggregation indices

Overdispersion can first be checked by fitting an intercept-only binomial GLM and testing for overdispersion.

```
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)

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 = 2.412
p-value = < 0.001
```

The C(alpha) test from the *epiphy* package tests the binomial null hypothesis directly.

```
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
```

Binary power law

The binary power law (BPL), introduced by [Hughes & Madden \(1992\)](#), relates the observed variance of disease incidence among sampling units to the variance expected for a binomial distribution (a random pattern). It is widely used to describe spatial heterogeneity in incidence data ([Madden et al., 2018; 2007c](#)). In its general form, $v = A_p v_{bin}^\beta$, where A_p and β are parameters. For a random (binomial) pattern, $A_p = 1$ and $\beta = 1$.

The BPL relates two variances:

- Observed Variance (V): $V = \text{var}[X] = E[(X - np)^2]$
- Expected Binomial Variance (V_{bin}): $V_{bin} = np(1 - p)$

For proportions:

- Observed Variance (v): $v = \text{var}[x] = E[(x - p)^2]$
- Expected Binomial Variance (v_{bin}): $v_{bin} = \frac{p(1-p)}{n}$

The binomial distribution assumes independent observations of X , often interpreted as a random spatial pattern. If X follows a binomial distribution, then $\beta = 1$ in the BPL equations. Deviations from this indicate spatial heterogeneity or overdispersion. The log-log transformation is often used in statistical analyses due to the linear relationship between the log-transformed variables. Interpretation is as follows: when $\beta = 1$: Spatial heterogeneity is constant, indicating fixed overdispersion; when $\beta > 1$: Overdispersion varies systematically with incidence, showing a change in spatial heterogeneity with disease incidence levels. In practice, researchers fit the log-log model to empirical data to estimate β and A_p . This involves ordinary least squares (OLS) regression:

$$\ln(v_j) = \ln(A_p) + \beta \ln \left[\frac{\hat{p}_j(1 - \hat{p}_j)}{n_j} \right] + \epsilon_j \quad (17.1)$$

where: - j refers to specific observations, - $\hat{p}$ is the estimated proportion, - ϵ_j is the residual error.

The intercept of the fitted model ($\ln(A_p)$) provides the scale parameter, and the slope (β) gives insight into the nature of spatial heterogeneity and overdispersion in the data.

PART IV

Epidemics and yield

18. Definitions and concepts

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.

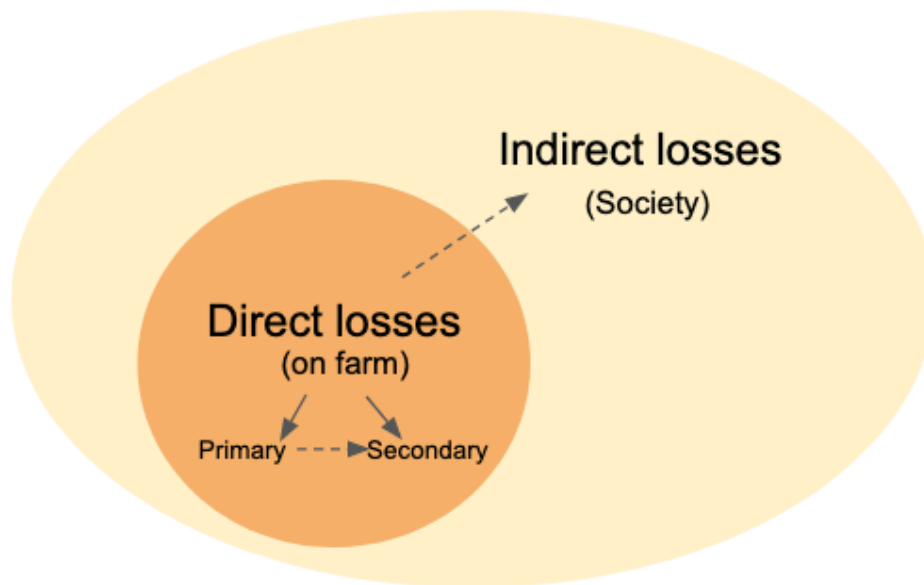


Figure 18.1: Typology of losses caused by plant diseases

Famous epidemics in history, such as the late blight of potatoes, serve as a reminder 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., 2007). 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).

Crop loss assessment

According to Madden et al. (2007), knowledge about the disease:yield relationship falls within crop loss assessment, a general branch of epidemiology that studies 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 is an upper limit that is rarely reached in practice; experimental plots managed with high inputs of fertilizers and pesticides approach it.
- **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)
y1 <- tibble::tribble(
  ~yield, ~value, ~class,
  "Theoretical", 25, 1,
  "Attainable", 20, 2,
  "Actual", 15, 3,
  "", 0, 4
)
y1 |>
  ggplot(aes(reorder(yield, -value), value, fill = class))+
  geom_col(width = 0.5)+
  r4pde::theme_r4pde(font_size = 14)+
  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 = "")
```

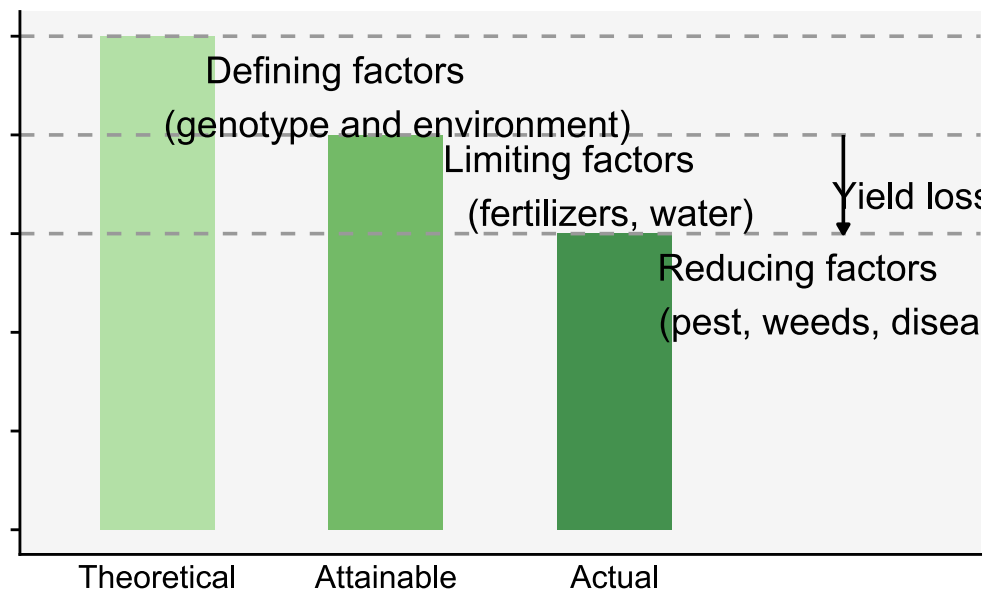


Figure 18.2: Yield levels

Generating disease:yield data

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:

- conducting experiments in controlled environments such as fields or greenhouses
- 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.

Damage curves

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., 2007). 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 (Madden et al., 2007).

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 were made freely available in this repository and were included in the package that accompanies the book.

```
library(tidyverse)
library(r4pde)
wm <- WhiteMoldSoybean

glimpse(wm, 60)
```

```
Rows: 382
Columns: 17
$ study      <dbl> 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9...
$ treat      <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,...
$ season     <chr> "2009/2010", "2009/2010", "2009/20...
$ harvest_year <dbl> 2010, 2010, 2010, 2010, 2010, 2010...
$ location   <chr> "Agua Fria", "Agua Fria", "Agua Fr...
$ state      <chr> "GO", "GO", "GO", "GO", "GO", "GO"...
$ country    <chr> "Brazil", "Brazil", "Brazil", "Bra...
$ elevation  <dbl> 891, 891, 891, 891, 891, 891, 891,...
$ region     <chr> "Northern", "Northern", "Northern"...
$ elevation_class <chr> "low", "low", "low", "low", "low",...
$ inc_check  <dbl> 37.7, 37.7, 37.7, 37.7, 37.7, 37.7...
$ inc_class  <chr> "low", "low", "low", "low", "low",...
$ yld_check  <dbl> 3729, 3729, 3729, 3729, 3729, 3729...
$ yld_class  <chr> "high", "high", "high", "high", "h...
$ inc        <dbl> 37.7, 11.6, 33.5, 1.0, 5.6, 1.0, 3...
$ scl        <dbl> 5092, 6154, 200, 180, 1123, 641, 1...
$ yld        <dbl> 3729, 3739, 3863, 3904, 4471, 4313...
```

As seen above, the full data set has 17 variables. Let's reduce the data set to a few variables of interest (study, inc and yld) and trials 1 to 4.

```
wm2 <- wm |>
  select(study, inc, yld) |>
  filter(study %in% c(1,2, 3, 4))
```

```
wm2
```

```
# A tibble: 52 × 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))+
  geom_point(size = 2)+
  theme_r4pde(font_size = 14)+
  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")

```

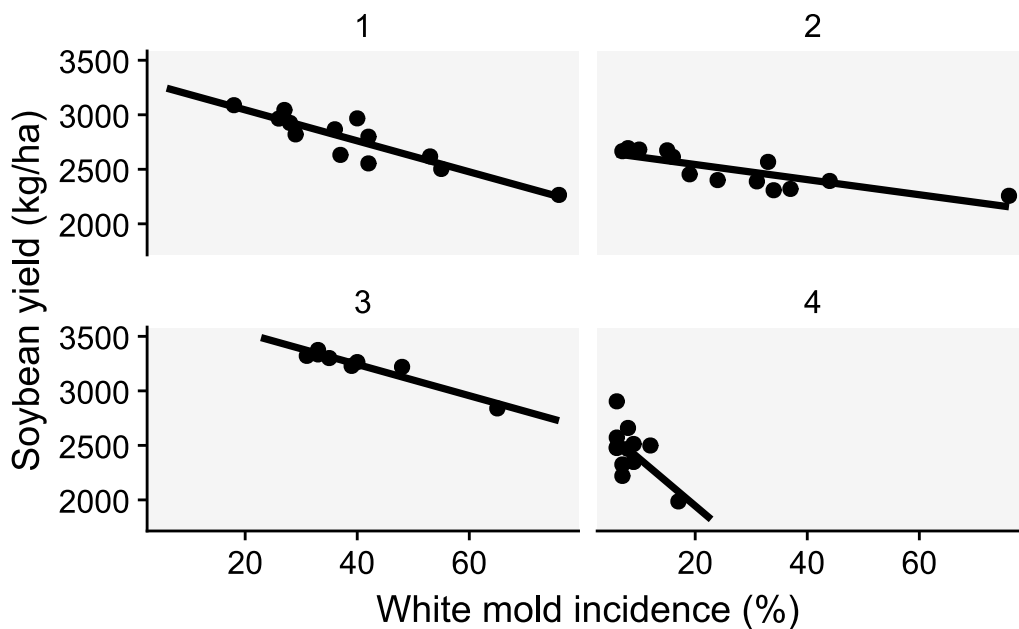


Figure 18.3: Relationship between soybean yield and incidence of white mold in four experiments (studies 1 to 4)

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_r4pde(font_size = 14)+
  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_r4pde(font_size = 14)+
  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
```

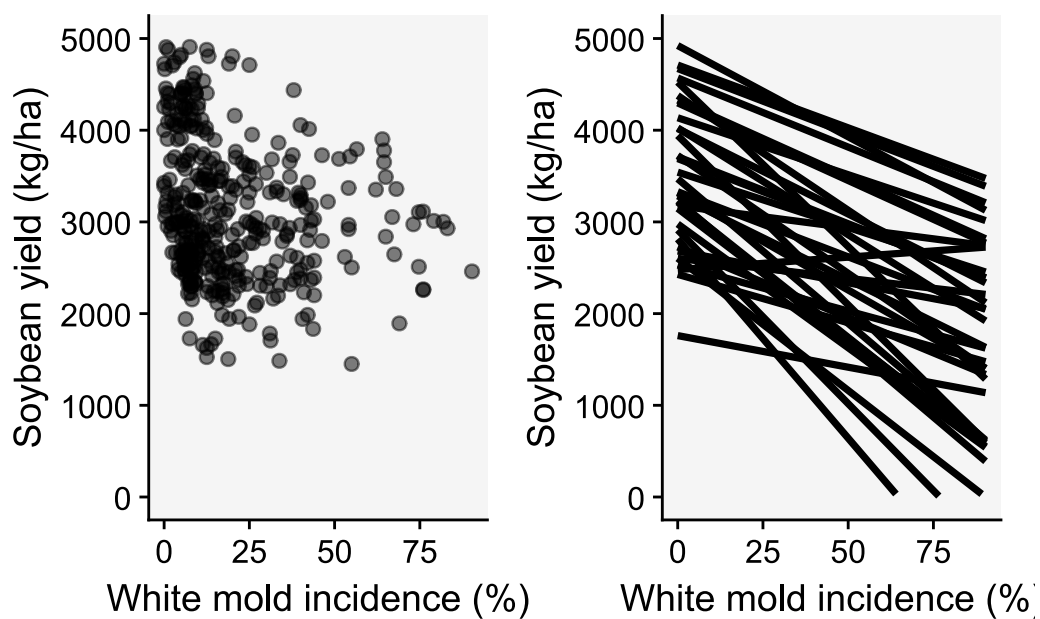


Figure 18.4: Relationship between soybean yield and incidence of white mold across trials. Observed (left) and fitted regression lines (right)

19. Statistical models

Introduction

When we aim to empirically describe the relationship between yield (either in absolute terms represented as [YLD], or in relative terms such as relative yield [RY] or relative yield loss [RL]) and disease intensity (denoted as y), the most straightforward approach is to employ a single metric of disease intensity. This could be in the form of disease incidence, severity, electronic measurements, among others. A linear model, as suggested by (Madden et al., 2007), often proves to be a suitable choice for this purpose. Such models are frequently termed as single-point or critical-point models. The nomenclature stems from the fact that the assessment of the disease occurs at one pivotal moment during the epidemic. This moment is typically chosen because it's when the disease's impact correlates with yield outcomes.

A version of the equation, for the absolute quantity of yield (YLD), is written as:

$$YLD = \beta_0 - \beta_1 y$$

where β_0 and β_1 are parameters and y is a disease measure. For this particular case, the intercept is a yield property of a given plant genotype in a given environment when the disease of interest is absent (or the attainable yield). The (negative) slope represents the change (reduction) in yield with change (increase) in disease intensity.

The interpretation of the model parameters depends on the specific yield variable being related to disease. If relative yield loss is related to disease, the slope will be positive, because the loss (relative decrease in yield) will increase with the increase in disease intensity assessed at the critical time. The differences in the intercept reflect the environmental effect on yield when disease is absent, while varying slopes reflect the environmental effect on yield response to disease (Madden et al., 2007).

Example 1: single study

We demonstrate the fitting of linear regression models (assuming a straight line relationship) to crop loss data on the effects of white mold on soybean yield (Lehner et al., 2016). This is the same data illustrated in the previous chapter. Let's load the data and assign them to a data frame named `wm`.

```
library(r4pde)
wm <- WhiteMoldSoybean
```

First, we will work with data from a single trial (trial 1).

```
library(tidyverse)
```

```
Warning: package 'ggplot2' was built under R version 4.5.3
```

Warning: package 'tidyr' was built under R version 4.5.3

Warning: package 'purrr' was built under R version 4.5.3

Warning: package 'dplyr' was built under R version 4.5.3

```
— Attaching core tidyverse packages — tidyverse 2.0.0 —
✓ dplyr      1.2.1    ✓ readr      2.1.5
✓ forcats    1.0.0    ✓ stringr    1.6.0
✓ ggplot2    4.0.3    ✓ tibble     3.3.0
✓ lubridate  1.9.4    ✓ tidyr      1.3.2
✓ purrr      1.2.2

— Conflicts — tidyverse_conflicts() —
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
wm1 <- wm |>
  dplyr::select(study, inc, yld) |>
  filter(study %in% c(1))
head(wm1, 13)
```

```
# A tibble: 13 × 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
11    1    27 3044
12    1    28 2925
13    1    36 2867
```

Linear regression

Assuming a linear relationship between `yld` and `inc`, we can employ a linear regression model for trial 1 using the `lm()` function.

```
lm1 <- lm(yld ~ inc, data = wm1)
jtools::summ(lm1)
```

Observations	13
Dependent variable	yld
Type	OLS linear regression

F(1,11)	46.86
R²	0.81
Adj. R²	0.79

	Est.	S.E.	t val.	p
(Intercept)	3329.14	86.84	38.33	0.00
inc	-14.21	2.08	-6.85	0.00

Standard errors: OLS

In the summary output, we can notice that the model explains a statistically significant and substantial proportion of variance. The model's intercept, corresponding to $\text{inc} = 0$, is at 3329.14. The effect of inc is statistically significant and negative ($\text{beta} = -14.21$ kg/ha per percentage point). In other words, 142.1 kg/ha is lost for each 10 percentage point increase in incidence, given the attainable yield of 3,329.14 kg/ha.

Damage coefficients

Damage curves offer a visual representation of how plant diseases can impact a given crop in terms of yield loss. When we want to normalize these effects to better compare across different systems or conditions, it is useful to express these curves in relative terms rather than absolute ones. To achieve this, we can adjust the derived slope by dividing it by the intercept. This step essentially scales the rate of damage in relation to the baseline or the starting point (when there's no damage). By subsequently multiplying the result by 100, we convert this value into a percentage. This percentage is termed the "relative damage coefficient". What makes this coefficient particularly useful is its ability to standardize the measurement of damage, facilitating comparisons across diverse pathosystems.

Two plots can be produced, one that shows the effect of the disease on the relative yield and the other on the effect on yield loss (which in this case represents a positive slope). Both representations can be found in the literature. Let's use the estimated coefficients and produce these two plots.

```
# Extract the coefficients from model fit
dc <- (lm1$coefficients[2]/lm1$coefficients[1])*100
dc
```

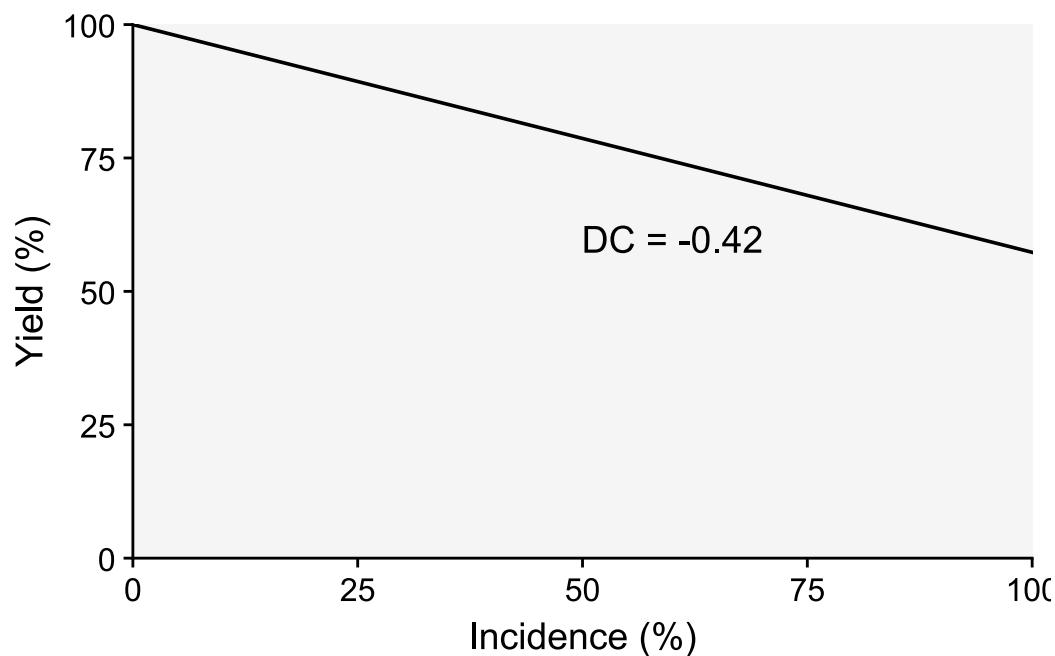
```
inc
-0.426775
```

```
# Plot the relative damage curve
x = seq(0,100,0.1)
y = seq(0,100,0.1)
dat <- data.frame(x,y)

p1 <- dat |>
  ggplot(aes(x,y))+
  theme_r4pde(font_size = 14)+
  geom_point(color = "NA")+
  scale_y_continuous(expand = c(0,0))+
```

```
scale_x_continuous(expand = c(0,0))+
geom_abline(aes(intercept = 100, slope = dc))+
labs(x = "Incidence (%)", y = "Yield (%)")+
annotate(geom = "text", x = 60, y = 60, label = "DC = -0.42")
```

p1

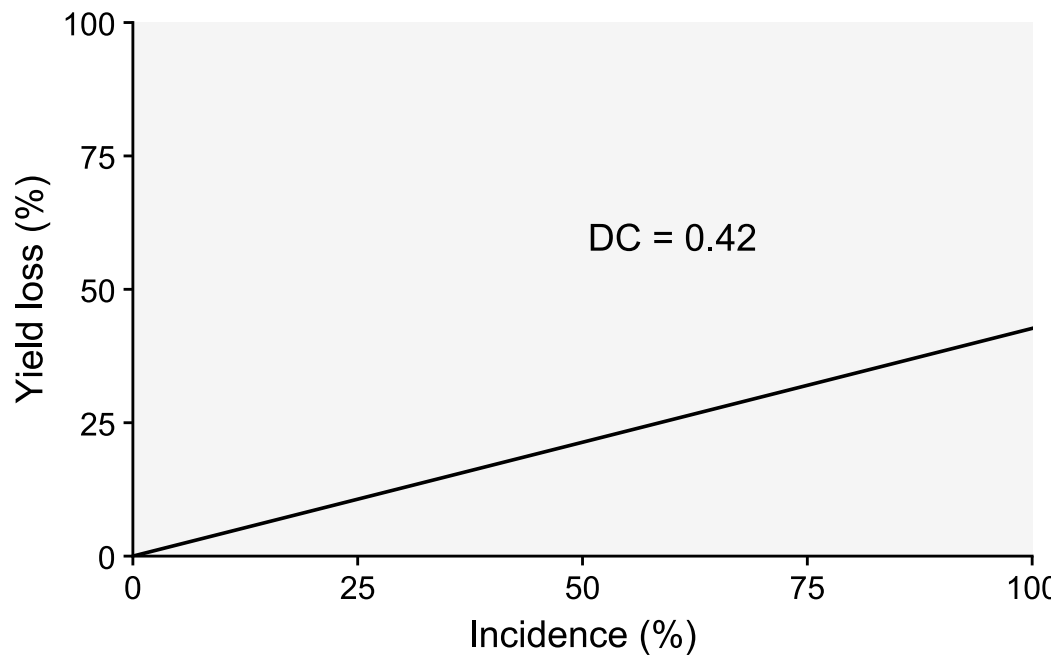


```
# Plot for the relative yield decrease
dc2 <- (-lm1$coefficients[2]/lm1$coefficients[1])*100
dc2
```

```
inc
0.426775
```

```
dat <- data.frame(x,y)
p2 <- dat |>
  ggplot(aes(x,y))+
  theme_r4pde(font_size = 14)+
  geom_point(color = "NA")+
  scale_y_continuous(expand = c(0,0))+
  scale_x_continuous(expand = c(0,0))+
  geom_abline(aes(intercept = 0, slope = dc2))+
  labs(x = "Incidence (%)", y = "Yield loss (%)")+
  annotate(geom = "text", x = 60, y = 60, label = "DC = 0.42")
```

p2



Example 2: Multiple studies

Introduction

When managing data sourced from multiple studies (or experiments), a naive and straightforward approach is to pool all the data and fit a “global” linear regression. This option, while efficient, operates under the assumption that the different studies share common underlying characteristics, which might not always be the case.

Another simplistic methodology entails running independent linear regressions for each trial and subsequently averaging the intercepts and slopes. While this provides a glimpse into the general behavior of the data, it potentially sidesteps important variations that exist between individual trials. This variation is crucial, as different trials could have unique environments, treatments, or experimental conditions, all of which can influence results.

Neglecting these variations can mask important nuances and lead to overgeneralized or inaccurate conclusions. In order to accommodate the inherent variability and structure present in multitrial data, researchers often turn to more refined analytical frameworks. Two such frameworks stand out in their ability to provide a nuanced view.

The first is a meta-analytic modelling. This approach synthesizes results across multiple studies to arrive at a more comprehensive understanding (Madden & Paul, 2011). In the context of linear regressions across multiple trials, the coefficients (both intercepts and slopes) from each trial can be treated as effect sizes (Dalla Lana et al., 2015). The standard error of these coefficients, which reflects the precision of the estimate, can then be used to weight each coefficient, ensuring that more reliable estimates have greater influence on the overall mean. This method can also provide insights into heterogeneity across trials, and if required, moderators can be explored to account for this variability (Lehner et al., 2016).

A second approach is to fit a random coefficients (mixed effects) models. This approach allows the intercepts and slopes to vary across different trials, treating them as random effects (Madden & Paul, 2009). By modeling the coefficients as random effects, we’re assuming they

come from a distribution, and we aim to estimate the parameters of this distribution. This structure acknowledges that variability exists between different trials and allows for the sharing of information across trials, thereby improving the estimation.

Both methods have their strengths and are appropriate under different circumstances. The choice largely depends on the research question, data structure, and the assumptions one is willing to make.

Global regression

Let's begin by fitting a "global regression", but we might want to first inspect the damage curve with the fit of the global regression model visually.

```
ggplot(wm, aes(inc, yld))+
  theme_r4pde(font_size = 12)+
  geom_point(shape = 1)+
  stat_smooth(method = lm, fullrange=TRUE, se = TRUE, col = "black")+
  ylab("Yield (kg/ha)")+
  xlab("White mold incidence (%)")
```

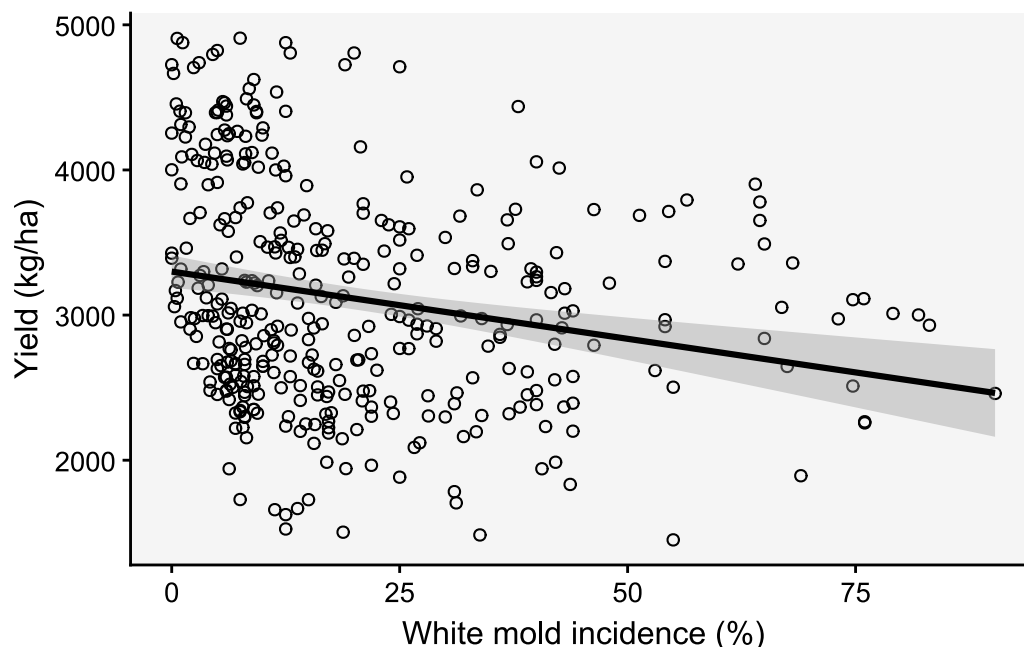


Figure 19.1: Soybean yield and white mold incidence across all trials, with the fitted global regression line and its 95% confidence band

A "global" regression can be performed using:

```
library(broom)
```

```
Attaching package: 'broom'
```

```
The following object is masked from 'package:r4pde':
```

```
augment
```

```
fit_global <- wm |>
  do(tidy(lm(.$yld ~ .$inc), conf.int=TRUE))
fit_global
```

```
# A tibble: 2 × 7
  term      estimate std.error statistic  p.value conf.low conf.high
  <chr>      <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
1 (Intercept) 3300.      56.5      58.5 5.14e-192 3189.    3411.
2 .$inc      -9.26      2.11     -4.39 1.45e- 5 -13.4    -5.12
```

The global intercept and slope were estimated as 3,299 kg/ha and -9.26 kg/p.p. (percent point), respectively.

Individual regressions

Now we can fit separate regressions for each trial.

```
ggplot(wm, aes(inc, yld, group = study))+
  theme_r4pde(font_size = 12)+
  geom_point(shape = 1)+
  stat_smooth(method = lm, se = F, col = "black")+
  ylab("Yield (kg/ha)")+
  xlab("White mold incidence (%)")
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
#facet_wrap(~ study, ncol = 7, scales = "fixed")
```

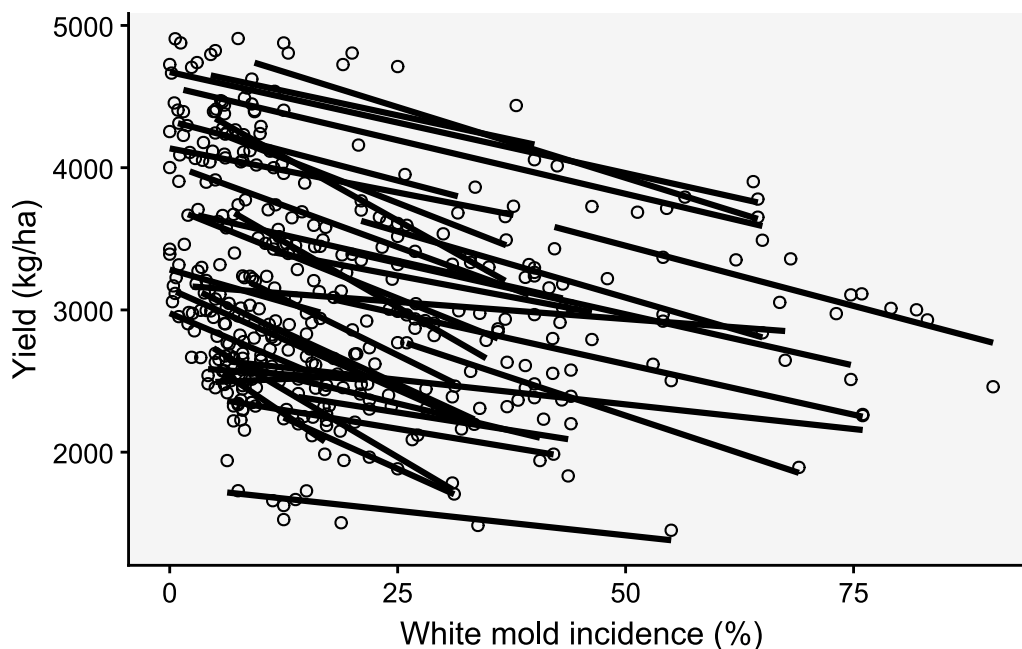


Figure 19.2: Soybean yield and white mold incidence, with a separate regression line for each trial

To fit separate regression lines for each study and extract the coefficients, we can use the `group_by()` function along with `do()` and of the `{dplyr}` package and `tidy()` from `{broom}` package.

The data is first grouped by the `study` column. Then, for each study, a linear regression is fitted with `yld` as the response variable and `inc` as the predictor. The `tidy` function from the `broom` package is then used to extract the coefficients and confidence intervals for each regression line. The resulting output should give us a *tidy* data frame with coefficients for each trial.

```
fit_all <- wm |>
  group_by(study) |>
  do(broom::tidy(lm(.yld ~ .$inc), conf.int=TRUE))
fit_all
```

```
# A tibble: 70 × 8
# Groups:   study [35]
  study term          estimate std.error statistic p.value conf.low conf.high
  <dbl> <chr>          <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
1     1 1 (Intercept)  3329.    86.8     38.3 4.60e-13  3138.    3520.
2     1 1 .$inc       -14.2     2.08    -6.85 2.78e- 5   -18.8    -9.64
3     2 2 (Intercept)  2682.    48.6     55.2 8.55e-15  2575.    2789.
4     2 2 .$inc        -6.93     1.49    -4.66 6.89e- 4    -10.2    -3.66
5     3 3 (Intercept)  4017.    61.6     65.2 1.37e-15  3882.    4153.
6     3 3 .$inc       -18.6     1.71   -10.9 3.11e- 7    -22.4   -14.9
7     4 4 (Intercept)  2814.   151.     18.6 1.15e- 9   2481.    3147.
8     4 4 .$inc      -43.5     16.8    -2.58 2.56e- 2    -80.5    -6.38
9     5 5 (Intercept)  3317.   234.     14.2 2.07e- 8   2802.    3832.
10    5 5 .$inc      -21.2     5.69    -3.72 3.36e- 3    -33.7    -8.67
# i 60 more rows
```

Now we can plot the distributions of each coefficient.

```
p3 <- fit_all |>
  filter(term == "(Intercept)") |>
  ggplot(aes(x = estimate))+
  geom_histogram(bins = 8, color = "white", fill = "gray50")+
  theme_r4pde()+
  labs(x = "Intercept", y = "Frequency")

p4 <- fit_all |>
  filter(term == ".$inc") |>
  ggplot(aes(x = estimate))+
  geom_histogram(bins = 8, color = "white", fill = "gray50")+
  theme_r4pde()+
  labs(x = "Slope", y = "Frequency")

library(patchwork)
```

Warning: package 'patchwork' was built under R version 4.5.3

p3 | p4

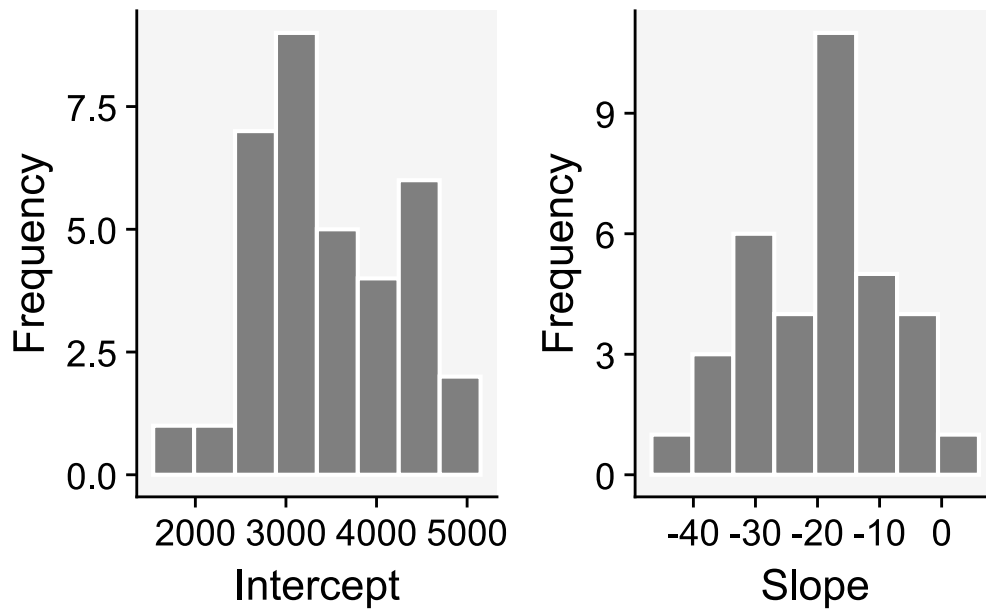


Figure 19.3: Distributions of the intercepts (left) and slopes (right) of the trial-specific regressions

Let's summarize the data on the slopes and intercept, using the `summary()` function.

```
fit_all |>
  filter(term == "(Intercept)") |>
  ungroup() |>
  dplyr::select(estimate) |>
  summary()
```

```
estimate
Min.   :1760
1st Qu.:2863
Median :3329
Mean    :3482
3rd Qu.:4080
Max.    :4923
```

```
fit_all |>
  filter(term == ".$inc") |>
  ungroup() |>
  dplyr::select(estimate) |>
  summary()
```

```
estimate
Min.   : -43.455
1st Qu.: -27.676
Median : -16.926
Mean    : -19.529
3rd Qu.: -13.054
Max.    :  2.712
```

Meta-analysis

Here the objective is to combine the estimates from multiple studies or trials into a single overall estimate using a random-effects meta-analysis using the {metafor} R package (Viechtbauer, 2010). The goal is to capture both the central tendency (i.e., the overall effect) and the variability (heterogeneity) among those individual estimates. Let's first prepare data for analysis and then run separate meta-analysis for the intercepts and slopes.

```
# data preparation
Intercepts <- fit_all |>
  filter(term == "(Intercept)")

Slopes <- fit_all |>
  filter(term == ".$inc")

# Model for the intercepts
library(metafor)
ma1 <- rma(yi = estimate, sei = std.error, data = Intercepts)
summary(ma1)
```

```
Random-Effects Model (k = 35; tau^2 estimator: REML)

      logLik   deviance      AIC      BIC     AICc
-274.9958   549.9916   553.9916   557.0444   554.3787

tau^2 (estimated amount of total heterogeneity): 607939.3750 (SE = 150941.5658)
tau (square root of estimated tau^2 value):      779.7047
I^2 (total heterogeneity / total variability):   98.88%
H^2 (total variability / sampling variability):   89.38

Test for Heterogeneity:
Q(df = 34) = 3402.9633, p-val < .0001

Model Results:

      estimate      se      zval      pval      ci.lb      ci.ub
3479.3087   133.3611   26.0894   <.0001   3217.9258   3740.6917   ***

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

```
# Model for the slopes
ma2 <- rma(yi = estimate, sei = std.error, data = Slopes)
summary(ma2)
```

```
Random-Effects Model (k = 35; tau^2 estimator: REML)

      logLik   deviance      AIC      BIC     AICc
-127.4587   254.9174   258.9174   261.9701   259.3045

tau^2 (estimated amount of total heterogeneity): 65.0917 (SE = 22.6013)
tau (square root of estimated tau^2 value):      8.0679
I^2 (total heterogeneity / total variability):   82.28%
```

```
H^2 (total variability / sampling variability): 5.64

Test for Heterogeneity:
Q(df = 34) = 151.3768, p-val < .0001

Model Results:

estimate      se      zval      pval      ci.lb      ci.ub
-18.1869  1.6648 -10.9245 <.0001 -21.4499 -14.9240 ***

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

In the output above, we can notice that there is considerable heterogeneity in both the intercepts and slopes ($P < 0.01$ for Q statistics), meaning that there may be moderator variables that, if added to the model, could potentially reduce the variance. The estimates for the intercept and slopes were 3479.30 kg/ha and -18.18 kg/p.p. (percentage point), respectively.

Random coefficients model

Among several options in the R ecosystem, the `{lme4}` package in R offers a comprehensive suite of tools to fit linear mixed-effects models (Bates et al., 2015). When analyzing data from multiple trials using this package, it allows for both intercepts and slopes to vary across the trials by treating them as random effects. By doing so, the inherent assumption is that these coefficients (intercepts and slopes) are drawn from certain distributions, and the goal becomes estimating the parameters of these distributions. This modeling technique acknowledges and captures the variability present across different trials. Importantly, by treating the coefficients as random effects, this enables the sharing of information across trials. This not only provides a more holistic understanding of the underlying data structure but also enhances the precision and robustness of the coefficient estimates.

The `lmer()` function allows us to fit a mixed model with both the fixed (inc) and random effects (inc | study).

```
library(lme4)
rc1 <- lmer(yld ~ inc + (inc | study), data = wm,
            REML = F)
summary(rc1)
```

```
Linear mixed model fit by maximum likelihood ['lmerMod']
Formula: yld ~ inc + (inc | study)
Data: wm

            AIC      BIC    logLik -2*log(L)  df.resid
5319.4    5343.1   -2653.7    5307.4      376

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.7078 -0.5991 -0.0295  0.5077  3.2364

Random effects:
Groups   Name              Variance Std.Dev. Corr
study    (Intercept) 557573.08  746.708
```

```

      inc      36.85   6.071  -0.29
Residual      37228.73 192.947
Number of obs: 382, groups: study, 35

Fixed effects:
      Estimate Std. Error t value
(Intercept) 3455.432    128.063   26.98
inc          -17.236     1.451  -11.88

Correlation of Fixed Effects:
      (Intr)
inc -0.300
optimizer (nloptwrap) convergence code: 0 (OK)
Model failed to converge with max|grad| = 0.416806 (tol = 0.002, component 1)
See ?lme4::convergence and ?lme4::troubleshooting.

```

Conclusion

Results from various regression methods show that the calculated damage coefficients fall within the range of -0.28 to -0.56 (see table below). This range of values represents the extent to which damage is influenced by the method in consideration. The most straightforward method, often referred to as the naive approach, produced the most conservative estimate, positioning the damage coefficient at the lower bound of the observed range. In stark contrast, computing the mean values from multiple individual regressions yielded a dc that topped the range, signifying a greater estimated impact.

Model	intercept	slope	damage coefficient
Global regression	3299.6	-9.261	-0.28
Mean of regressions	3482	-19.529	-0.56
meta-analysis	3479.3	-18.1869	-0.52
mixed-models	3455.43	-17.236	-0.49

However, it's worth noting that the coefficients derived from the more advanced techniques - meta-analysis and mixed-models - were quite congruent. Their close alignment suggests that both methodologies, while operating on different principles, capture the underlying dynamics of the data in somewhat analogous ways. A prominent advantage of employing these advanced techniques is their ability to encapsulate and quantify uncertainty. This capability is crucial in scientific analyses as it provides a clearer understanding of the confidence levels associated with the derived coefficients. By being able to measure and articulate this uncertainty, researchers can ensure their interpretations and subsequent decisions are founded on a solid empirical base.

PART V

Disease prediction

20. Warning systems

Introduction

One practical application of plant disease epidemiology is to predict disease occurrences to guide timely management interventions, reducing crop damage and rationalizing pesticide use. Since the early to mid-1900s, warning systems (synonyms: disease forecaster, predictor) have advanced considerably. More comprehensively, these systems have evolved to a Decision Support System (DSS), when they integrate expert input, models, and databases for more nuanced management recommendations, transcending simple prediction to encompass various goals within computerized frameworks. In fact, technological progress in recent decades has boosted the development and automation of DSSs, now widely available in public and private sectors, providing direct, sophisticated guidance to crop advisors and growers (Figure 20.1).

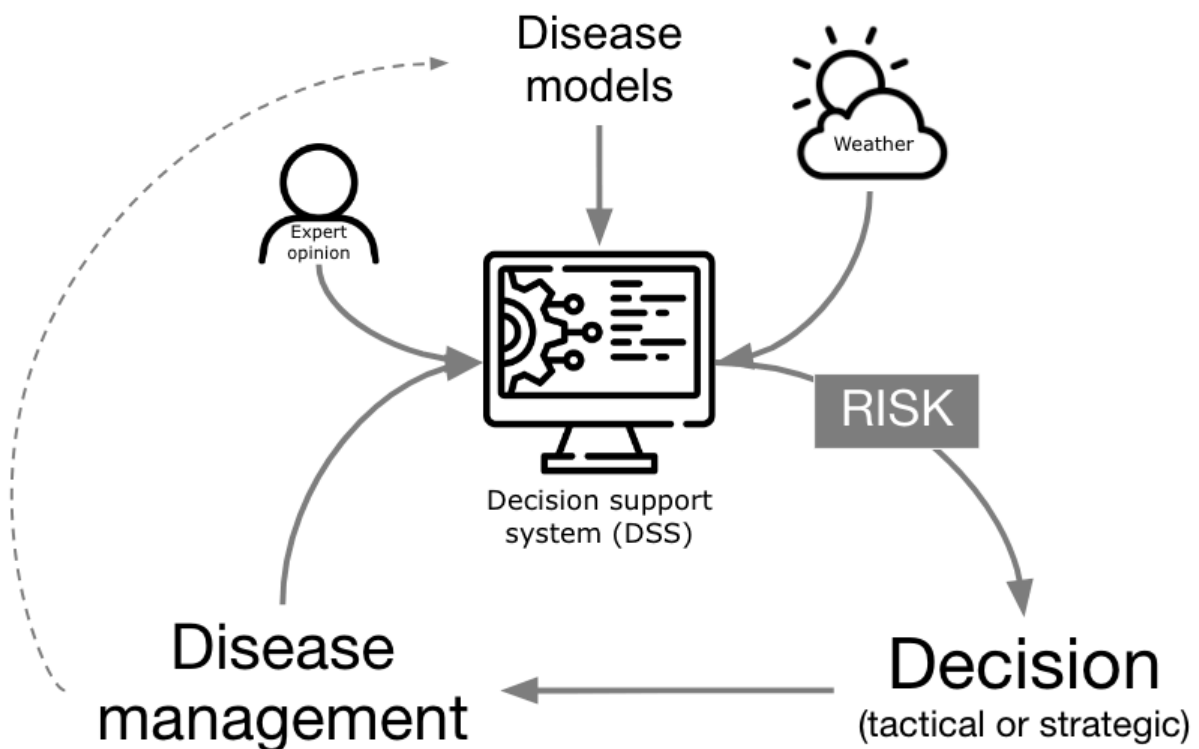


Figure 20.1: Core elements of a decision support system that provides risk information for plant disease management

At its core, a DSS targeting risk prediction for plant disease management is based on a disease **model**, or a simplified, often mathematical, representation of a system (here the pathosystem) used for making predictions or suggesting management decisions based on the risk

information they provide. The models range from basic rules (e.g. if-then) and static thresholds to sophisticated simulation models covering entire disease epidemics. Historically, these models collect weather information from on site automatic weather stations, but can be fed with data from remote sensing (e.g. satellite) or data reanalysis sources.

Risk assessment and decision framework

There are basically two types of decisions related to plant disease management: tactical and strategic, and these can be related to the distinct time frames of information (i.e., historical data, pre-season, growing season, and future seasons). A risk assessment and decision framework, with associated terminology, can be proposed based on such relations (Figure 20.2) and its components are explained next.

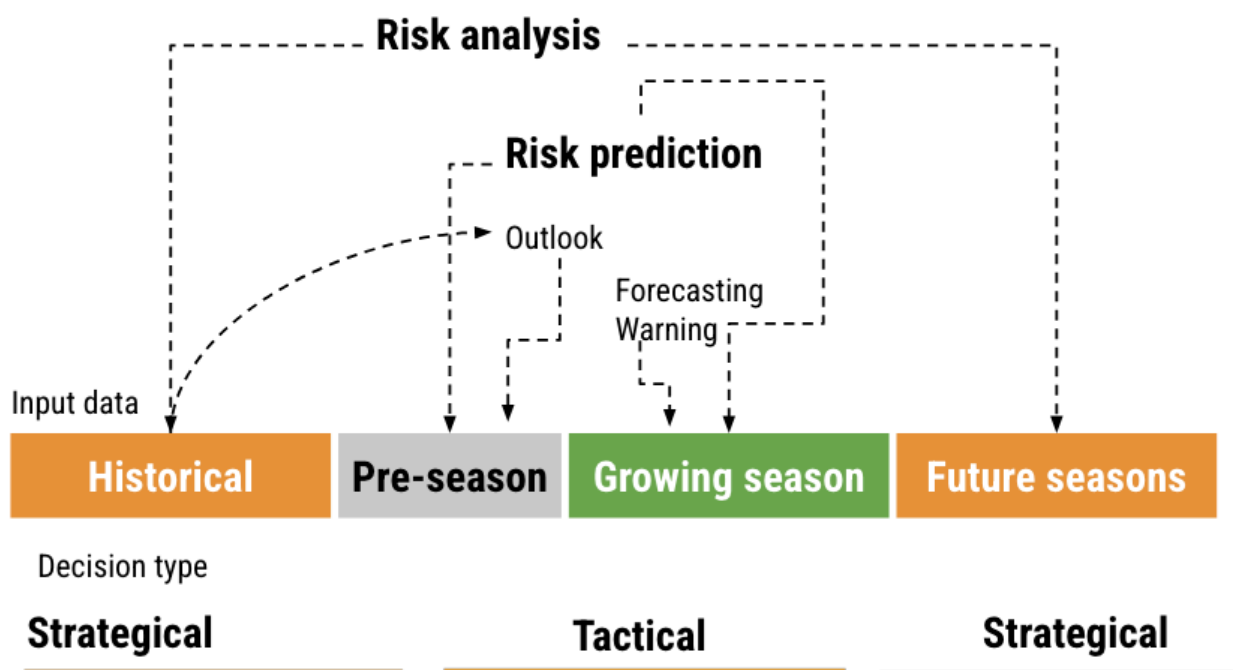


Figure 20.2: Framework and associated terminology for plant disease risk assessment

Historical data and prediction refers to those obtained from or simulated for previous years or seasons. This can include past observed weather patterns, observed or simulated disease outbreaks. The goal is to learn about the “normal” patterns associated with disease prevalence and severity. In the **pre-season** period, predictions are made before the actual planting or growing season starts. This could be based on early predictions of weather patterns for the season. During the **growing season**, real-time data is used to provide short-term predictions. This is the period when crops are in the fields and are actively monitored. Finally, for **future seasons**, projections (e.g. climate change scenarios) can be made for the subsequent years or decades.

Strategic decisions are those with a long-term impact and are typically based on historical data and projections for future seasons. Examples might include choosing where to plant the crops in the next years, which cultivar to plant, and making infrastructure investments.

Tactical are short-term decisions typically based on real-time data or short-term predictions,

especially those relevant to the current growing season. A chief example includes the timing of fungicide applications.

As to risk terminology, **risk analysis** is a comprehensive assessment of potential disease risks, considering both historical data and projections of the future. The goal is to understand and mitigate potential threats to agricultural production. **Risk prediction** can be broken down into a) **outlook**: Broad predictions or estimations about potential risks, often based on pre-season data. Example: effects that El Niño Southern Oscillation (ENSO) may have on disease patterns; b) **forecasting**: short-term predictions, typically for the growing season, such as critical weather events; and c) **warning**: Immediate alerts about imminent risks, like an upcoming risk of plant infection, crucial for tactical decisions.

In this chapter we will focus on warning systems that make use of disease monitoring or seasonal weather to provide risk information for tactical decisions.

When is a warning system needed?

The figure and the box below provide some information on their utility. In the figure, risk analysis is defined as an approach for pathogens/diseases that are not present in a target region and for which modeling can be used for risk estimation if the disease is highly damaging where it occurs. Warning systems can be used if the disease is more erratic but still damaging when occurring. If the epidemics are too frequent and no effective and economic control is available (for example, some nematodes and viruses), plant host resistance is the way to go. If there are effective and economic control measures (e.g. fungicide sprays) during the season, farmers should follow scheduled application of treatments.

i Interactive content

The decision chart for the need of risk analysis, warning systems, scheduled treatments or host resistance is interactive and only available in the online edition of the book: <https://r4pde.net/prediction-warning-systems.html>.

⚠ When is a warning system useful?

For operational and economic use, warning systems must fulfill four criteria to be practical. A more comprehensive discussion on this topic is presented in (Campbell & Madden, L., 1990):

1. The targeted disease must be economically damaging to crop yield or quality. Minor diseases with negligible effects on yield or quality are unlikely to gain attention from growers and advisors.
2. The diseases should exhibit variability across seasons in terms of onset, epidemic growth rate, severity, or another aspect, creating uncertainty in decision-making. Diseases with predictable patterns provide minimal information and little management advantage, reducing the relevance of warning systems.
3. Users should be capable of acting on the system's alerts, necessitating available and effective control measures and sufficient response time to prevent crop damage. Systems are unhelpful if practitioners can't adapt their strategies promptly.
4. The system must encompass comprehensive knowledge about the disease, synthesizing accurate risk estimates. Understanding the specific interactions between host, pathogen, and environment is crucial for the system's effectiveness and relevance.

What types of systems are there?

These systems vary significantly in structure and design, reflecting the multitude of plant diseases, objectives, available data, control strategies, developer preferences, and operational infrastructures. Usually, warning systems are based on weather inputs, but they might leverage other inputs like host, pathogen, and economic factors, catering to the complexities of disease prediction.

Disease warning systems can include static or dynamic disease thresholds, direct detection of inoculum, simple rules of thumb (e.g. if-then) based on weather, infection risk during defined periods (risk models), or complex simulation models that estimate all phases of an epidemic. Let's see some examples of these systems together with a possible implementation in R.

Disease thresholds

Damaging thresholds, integral to integrated pest management in entomology, can serve as a basic disease warning system. They involve economic injury levels, denoting pest abundance that equates control costs with incurred losses, and economic or action thresholds, indicating when action is necessary to avoid reaching injury levels (Pedigo et al., 1986). These concepts, while straightforward, can be quite complex in practical scenarios.

Though less prevalent than in arthropod management, thresholds guide actions like fungicide application in plant diseases, especially those directly impacting yield through photosynthetic area reduction (Leiminger & Hausladen, 2012; Nutter et al., 1993). However, their application is challenging for rapid, recurrent diseases affecting high-value crops, requiring prompt intervention even at minimal disease levels. With potato late blight, for instance, the first fungicide application may need to be applied by the time disease severity reaches as low as 0.1% of the foliage. Hence, disease monitoring for detection and quantification is vital for this system. In reality, for some rapidly spreading and highly damaging diseases one cannot wait to

“see” the disease to start protecting the crops, for which yield protection is best when applications are made preventively.

However, the concept of **economic damage threshold** (EDT) may be used as a criterion to indicate when to start with fungicide sprays. By definition, EDT is the amount of disease intensity (e.g. severity when dealing with foliar diseases) that corresponds to an economic loss that equates the control cost to combat the disease. A formula for the EDT was proposed by Mumford and Norton (Mumford & Norton, 1984) and further modified by Reis (Reis et al., 2002) for use in foliar fungal diseases, as described below:

$$EDT = \frac{F_C}{C_P \cdot D_C} \cdot C_e$$

where EDT is the disease intensity, F_C is the fungicide cost (USD/ha), C_P is the crop selling price (USD/ton), D_C is the damage coefficient (adjusted to potential yield) and C_e is the control efficacy of the fungicide (proportion of disease reduction relative to non-treated). In practice, sprays should be applied prior to reaching the EDT, which gives rise to the ADT (action damage threshold).

In a study on northern corn leaf blight in Argentina, the following values were used to calculate the EDT (De Rossi et al., 2022). Note that the authors adjusted the Dc to potential yield by multiplying by the potential yield value (8.5 t/ha) in metric tons, since the Dc is normalized to one metric ton. More about Dc in the dedicated chapter on **yield loss**. The action damage threshold (ADT) was defined in that study as 20% reduction of the EDT.

```
calculate_EDT <- function(Fc, Cp, Dc, Ce) {
  EDT <- (Fc / (Cp * Dc)) * Ce
  return(EDT)
}
Fc <- 30      # fixed cost of control is 30 USD/ha.
Cp <- 112     # fixed crop price is 112 USD/ton.
Dc <- 0.1712  # for potential yield of 8.5 t/ha so 8.5 x 0.02015 = 0.1712.
Ce <- 0.70    # control efficacy of fungicide is 70%.
EDT_value <- calculate_EDT(Fc, Cp, Dc, Ce)
print(EDT_value)
```

```
[1] 1.09521
```

```
ADT = EDT_value * 0.80
ADT
```

```
[1] 0.8761682
```

As we can see, the EDT is variable depending on these four parameters. But how does the EDT vary according to variations on each of the parameters individually? Let's perform some simulations with varying values of each parameter.

```
library(tidyverse)
library(r4pde)

# The function is vectorized, so each parameter is varied while the
# others are kept at the baseline values defined above

# Vary crop price
```

```

Cp_values <- seq(100, 500, by = 5)
p_cp <- data.frame(Cp_values,
                  EDT_values = calculate_EDT(Fc, Cp_values, Dc, Ce)) |>
  ggplot(aes(Cp_values, EDT_values))+
  geom_line()+
  labs(x = "Crop price (US$/ton)",
       y = "EDT")+
  theme_r4pde(font_size =14)

# Vary fungicide (control) cost
Fc_values <- seq(10, 100, by = 5)
p_fc <- data.frame(Fc_values,
                  EDT_values = calculate_EDT(Fc_values, Cp, Dc, Ce)) |>
  ggplot(aes(Fc_values, EDT_values))+
  geom_line()+
  labs(x = "Control cost (US$/ha)",
       y = "EDT")+
  theme_r4pde(font_size =14)

# Vary damage coefficient
Dc_values <- seq(0.05, 0.30, by = 0.005)
p_dc <- data.frame(Dc_values,
                  EDT_values = calculate_EDT(Fc, Cp, Dc_values, Ce)) |>
  ggplot(aes(Dc_values, EDT_values))+
  geom_line()+
  labs(x = "Damage coefficient",
       y = "EDT")+
  theme_r4pde(font_size =14)

# Vary control efficacy
Ce_values <- seq(0.2, 1, by = 0.01)
p_ce <- data.frame(Ce_values,
                  EDT_values = calculate_EDT(Fc, Cp, Dc, Ce_values)) |>
  ggplot(aes(Ce_values, EDT_values))+
  geom_line()+
  labs(x = "Control efficacy (proportion)",
       y = "EDT")+
  theme_r4pde(font_size =14)

```

```

library(patchwork)
(p_cp | p_fc) /
  (p_dc | p_ce)

```

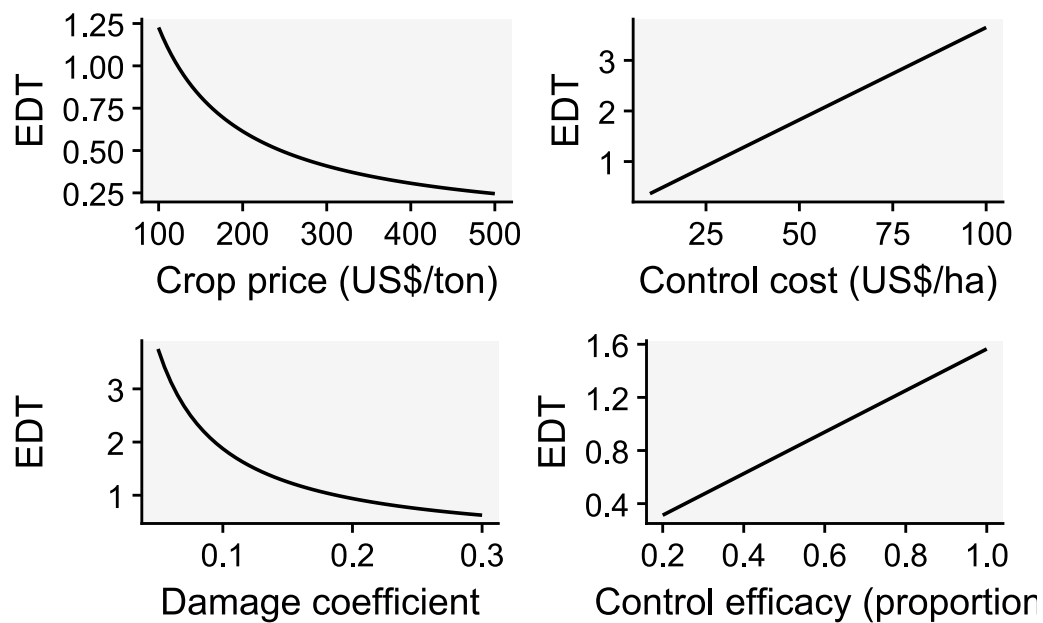


Figure 20.3: Variation of the economic damage threshold (EDT) according to fluctuations on each parameter in the formula for its calculation

Bonus app: EDT calculator

The scenario for EDT calculation is intricate, influenced by fluctuations in several parameters. To streamline these computations, a Shiny application called **EDT Calculator** was developed, enabling users to specify each parameter and assess EDT (and respective 95% CI) under uncertainties in variables such as crop price and fungicide costs.

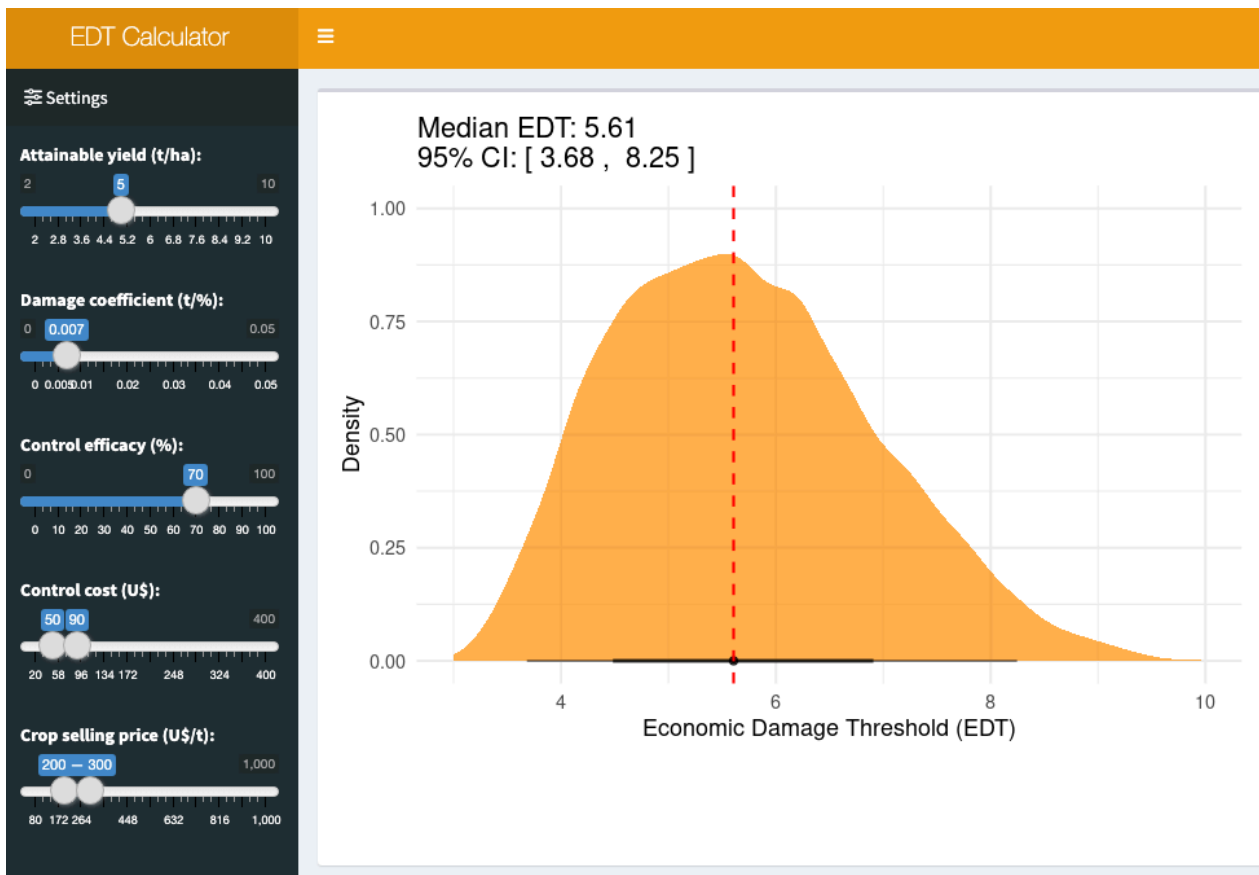


Figure 20.4: Screenshot of the EDT calculator, a Shiny app for interactive calculation of economic damage threshold (EDT)

Monitoring season-long weather

Disease warning systems frequently predict conditions conducive to infection of the plant by the pathogen, with wetness and temperature being key variables for many foliar diseases (Bourke, 1970). BLITECAST, the first computerized system (Krause et al., 1975) (which combined the Wallin and Hyre systems) provided the means of performing necessary calculations accurately and quickly and issuing recommendations to growers, is an example of successful automated warning system (Krause & Massie, 1975). While initial inoculum is often undetectable, the presence of inoculum of the pathogen is assumed in many weather-based warning systems. Predictions of an outbreak are possible by tracking environmental conditions favorable for disease development.

Weather-based disease warning systems, like FAST (Forecasting *Alternaria solani* on tomatoes) and Wallin (Madden, 1978; Wallin, 1962), continuously monitor moisture and temperature for various crop diseases. These systems calculate weather favorability from environmental data, predicting infection and disease severity. They serve to guide growers on optimal spraying schedules, initiating treatments or determining application intervals based on accumulated severity values over time.

Wallin model

J.R. Wallin developed a model in the mid-20th century focusing on forecasting potato late blight, detailed across several publications (Wallin, 1962). The model tracks hourly relative humidity and temperature, emphasizing periods with relative humidity of 90% or more. It calculates the number of high-humidity hours and the corresponding average temperature

during the wet period. By accumulating 'disease severity values' (DSV) from plant emergence throughout the season, based on humidity and temperature measures, the model predicts the initial onset and subsequent spread of potato late blight. The table below summarizes the way the DSVs are obtained based on combinations of hours of relative humidity > 90% and the air temperature within the wet period.

Table. Relationship of temperature and relative humidity (RH) periods as used in the Wallin late blight forecasting system to predict disease severity values (0 to 4).

	Daily severity value				
Average Temperature Range	0	1	2	3	4
7.2 - 11.6 C	15	16-18	19-21	22-24	>25
11.7 - 15.0 C	12	13-15	16-18	19-21	>22
15.1 - 26.6 C	9	10-12	13-15	16-18	>19

Let's calculate DSV in R based on Wallin's system. But first we need to download hourly weather data from NASA POWER project using {nasapower} R package for the locality of Viçosa, MG, Brazil during the month of March 2022.

```
library(nasapower)
weather <- get_power(
  community = "ag",
  lonlat = c(-42.88, -20.7561),
  pars = c("RH2M", "T2M"),
  dates = c("2022-03-02", "2022-03-31"),
  temporal_api = "hourly"
)
head(weather)
```

```
# A tibble: 6 × 8
  LON  LAT  YEAR  MO  DY  HR  RH2M  T2M
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 -42.9 -20.8 2022   3   2   0  96.3  19.4
2 -42.9 -20.8 2022   3   2   1  97.2  19.0
3 -42.9 -20.8 2022   3   2   2  98.2  18.8
4 -42.9 -20.8 2022   3   2   3  99.1  18.6
5 -42.9 -20.8 2022   3   2   4  99.0  18.6
6 -42.9 -20.8 2022   3   2   5  98.6  18.6
```

We now need to obtain the wet period (let's call it leaf wetness, or LW) based on hours of relative humidity >90% and then the average temperature during the LW period for each day. We can obtain these by grouping the variables by year, month and day and use `mutate()` and `summarise()`.

```
library(tidyverse)
weather2 <- weather |>
  group_by(YEAR, MO, DY) |>
```

```
mutate(LW = case_when(RH2M > 90 ~ 1,
                     TRUE ~ 0)) |>
filter(LW > 0) |>
summarise(Air_LWD = mean(T2M, na.rm = TRUE),
          LWD = n())
```

Now we are ready to calculate the daily DSV based on Wallin's rules on the table and inspect the first 6 rows of the new table called `df_wallin`.

```
df_wallin <- weather2 |>
mutate(
  DSV = case_when(
    # Temperature Range: 7.2 - 11.6 C
    Air_LWD >= 7.2 & Air_LWD <= 11.7 & LWD <= 15 ~ 0,
    Air_LWD >= 7.2 & Air_LWD <= 11.7 & LWD > 15 & LWD <= 18 ~ 1,
    Air_LWD >= 7.2 & Air_LWD <= 11.7 & LWD > 18 & LWD <= 21 ~ 2,
    Air_LWD >= 7.2 & Air_LWD <= 11.7 & LWD > 21 & LWD <= 24 ~ 3,
    Air_LWD >= 7.2 & Air_LWD <= 11.7 & LWD > 24 ~ 4,

    # Temperature Range: 11.7 - 15.0 C
    Air_LWD > 11.7 & Air_LWD <= 15.1 & LWD <= 12 ~ 0,
    Air_LWD > 11.7 & Air_LWD <= 15.1 & LWD > 12 & LWD <= 15 ~ 1,
    Air_LWD > 11.7 & Air_LWD <= 15.1 & LWD > 15 & LWD <= 18 ~ 2,
    Air_LWD > 11.7 & Air_LWD <= 15.1 & LWD > 18 & LWD <= 21 ~ 3,
    Air_LWD > 11.7 & Air_LWD <= 15.1 & LWD > 21 ~ 4,

    # Temperature Range: 15.1 - 26.6 C
    Air_LWD > 15.1 & Air_LWD <= 26.6 & LWD <= 9 ~ 0,
    Air_LWD > 15.1 & Air_LWD <= 26.6 & LWD > 9 & LWD <= 12 ~ 1,
    Air_LWD > 15.1 & Air_LWD <= 26.6 & LWD > 12 & LWD <= 15 ~ 2,
    Air_LWD > 15.1 & Air_LWD <= 26.6 & LWD > 15 & LWD <= 18 ~ 3,
    Air_LWD > 15.1 & Air_LWD <= 26.6 & LWD > 18 ~ 4,

    # Default (For temperatures out of the specified ranges or any other scenarios)
    TRUE ~ 0 # Assigning a default value of 0
  )
)
head(df_wallin)
```

```
# A tibble: 6 × 6
# Groups:   YEAR, MO [1]
  YEAR    MO    DY Air_LWD    LWD    DSV
<dbl> <dbl> <dbl>   <dbl> <int> <dbl>
1  2022     3     2    19.2     10     1
2  2022     3     3    19.7     12     1
3  2022     3     4    19.9     10     1
4  2022     3     5    19.5      7     0
5  2022     3     6    19.0      9     0
6  2022     3     7    18.4      7     0
```

We can visualize the daily and cumulative DSV for the monthly period after transforming the date format using `as.Date()` function. The dashed horizontal line in the plot indicates the action threshold of 20 cumulative DSV points, or when a spray should be applied. Please note that in real systems, the DSV is reduced to zero and another DSV counting is initiated after the spray.

```
df_wallin2 <- df_wallin |>
  mutate(DSV2 = cumsum(DSV),
         date = as.Date(sprintf('%04d-%02d-%02d', YEAR, MO, DY)))

df_wallin2 |>
  ggplot(aes(date, DSV))+
  geom_col(fill = "darkred")+
  geom_line(aes(date, DSV2))+
  geom_hline(yintercept = 20, linetype = 2)+
  annotate(geom = "text", x = as.Date("2022-03-04"), y = 20.5, label = "Action threshold")+
  r4pde::theme_r4pde()+
  labs(x = "Date", y = "Daily and cumulative DSV")
```

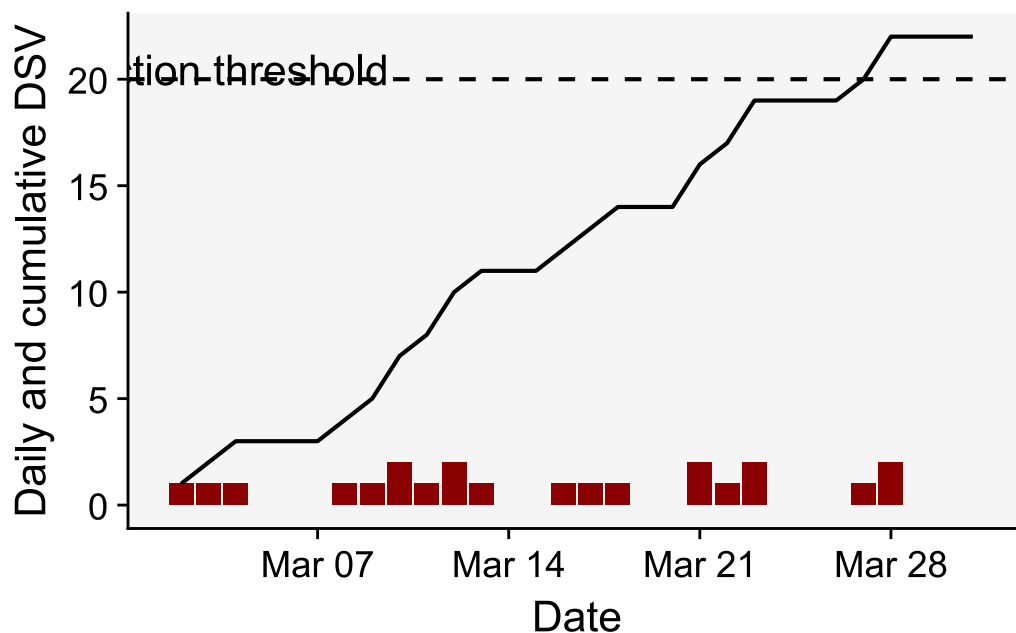


Figure 20.5: Daily (bars) and cumulative (line) disease severity values (DSV) calculated with Wallin's rules; the dashed line is the action threshold of 20 DSV

FAST model

Here is a code to calculate daily DSV values based on the FAST table (Madden, 1978) that relates the hours of relative humidity > 90% (wet period) and temperature during the wet period during a 24-hour period.

Mean temp (°C)	0	1	2	3	4
13-17	0-6	7-15	16-20	21+	23+
18-20	0-3	4-8	9-15	16-22	23+
21-25	0-2	3-5	6-12	13-20	21+
26-29	0-3	4-8	9-15	16-22	23+

```
df_fast <- weather2 |>
  mutate(
    DSV = case_when(
      # Temperature Range: 13 <= T < 18
```

```

Air_LWD >= 13 & Air_LWD < 18 & LWD >= 0 & LWD <= 6 ~ 0,
Air_LWD >= 13 & Air_LWD < 18 & LWD >= 7 & LWD <= 15 ~ 1,
Air_LWD >= 13 & Air_LWD < 18 & LWD >= 16 & LWD <= 20 ~ 2,
Air_LWD >= 13 & Air_LWD < 18 & LWD > 20 ~ 3,

# Temperature Range: 18 <= T < 21
Air_LWD >= 18 & Air_LWD < 21 & LWD >= 0 & LWD <= 3 ~ 0,
Air_LWD >= 18 & Air_LWD < 21 & LWD >= 4 & LWD <= 8 ~ 1,
Air_LWD >= 18 & Air_LWD < 21 & LWD >= 9 & LWD <= 15 ~ 2,
Air_LWD >= 18 & Air_LWD < 21 & LWD >= 16 & LWD <= 22 ~ 3,
Air_LWD >= 18 & Air_LWD < 21 & LWD > 22 ~ 4,

# Temperature Range: 21 <= T < 26
Air_LWD >= 21 & Air_LWD < 26 & LWD >= 0 & LWD <= 2 ~ 0,
Air_LWD >= 21 & Air_LWD < 26 & LWD >= 3 & LWD <= 5 ~ 1,
Air_LWD >= 21 & Air_LWD < 26 & LWD >= 6 & LWD <= 12 ~ 2,
Air_LWD >= 21 & Air_LWD < 26 & LWD >= 13 & LWD <= 20 ~ 3,
Air_LWD >= 21 & Air_LWD < 26 & LWD > 20 ~ 4,

# Temperature Range: 26 <= T < 30
Air_LWD >= 26 & Air_LWD < 30 & LWD >= 0 & LWD <= 3 ~ 0,
Air_LWD >= 26 & Air_LWD < 30 & LWD >= 4 & LWD <= 8 ~ 1,
Air_LWD >= 26 & Air_LWD < 30 & LWD >= 9 & LWD <= 15 ~ 2,
Air_LWD >= 26 & Air_LWD < 30 & LWD >= 16 & LWD <= 22 ~ 3,
Air_LWD >= 26 & Air_LWD < 30 & LWD > 22 ~ 4,

# Default (For temperatures out of the specified ranges or any other scenarios)
TRUE ~ 0 # Assigning a default value of 0
)
)
df_fast

```

```

# A tibble: 30 × 6
# Groups:   YEAR, MO [1]
  YEAR    MO    DY Air_LWD    LWD    DSV
  <dbl> <dbl> <dbl>   <dbl> <int> <dbl>
1  2022     3     2    19.2     10     2
2  2022     3     3    19.7     12     2
3  2022     3     4    19.9     10     2
4  2022     3     5    19.5      7     1
5  2022     3     6    19.0      9     2
6  2022     3     7    18.4      7     1
7  2022     3     8    19.2     10     2
8  2022     3     9    19.6     12     2
9  2022     3    10    19.6     13     2
10 2022     3    11    19.7     12     2
# i 20 more rows

```

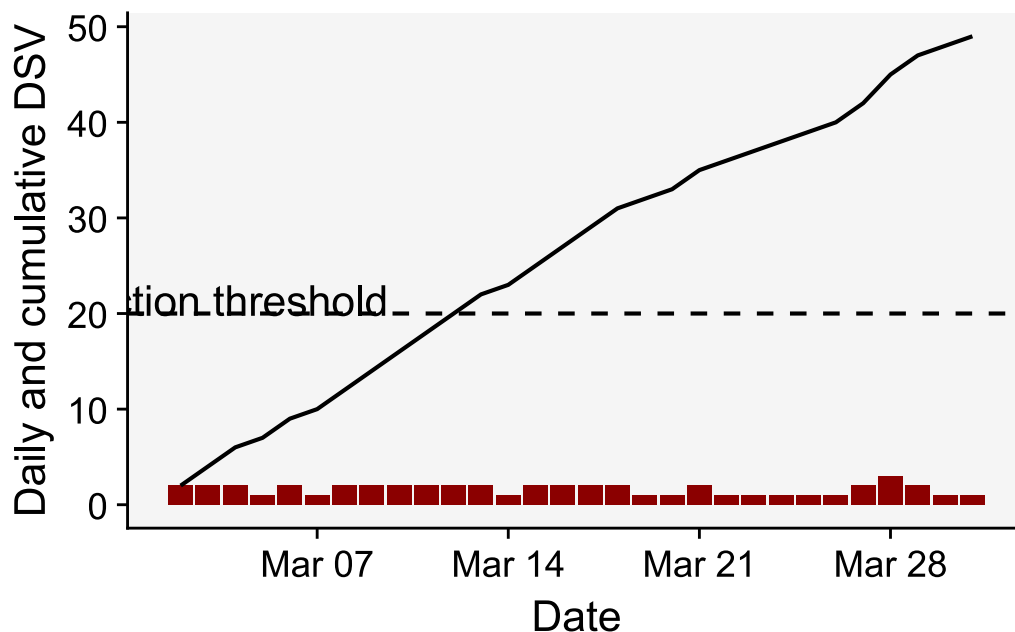
```

df_fast2 <- df_fast |>
  mutate(DSV2 = cumsum(DSV),
         date = as.Date(sprintf('%04d-%02d-%02d', YEAR, MO, DY)))

df_fast2 |>
  ggplot(aes(date, DSV))+
  geom_col(fill = "darkred")+
  geom_line(aes(date, DSV2))+

```

```
geom_hline(yintercept = 20, linetype = 2)+
annotate(geom = "text", x = as.Date("2022-03-04"), y = 21.5, label = "Action threshold")+
r4pde::theme_r4pde()+
labs(x = "Date", y = "Daily and cumulative DSV")
```



Bonus app: Dashboard

For educational purposes, an interactive dashboard was developed using R Shiny. This app demonstrates the application of the Wallin (forecast for late blight of potato) and FAST (Forecast for *Alternaria solani* on Tomatoes) rules for calculating DSV, determining the appropriate timing for fungicide sprays (based on a defined threshold), and counting the total sprays during a selected period (Figure 20.6).

To utilize the system, users should select the model, input the latitude and longitude (or choose a location from the map), and specify the time period for the simulation, such as from plant emergence to harvest. The weather data for this simulation is sourced from the NASA POWER project via the {nasapower} R package.

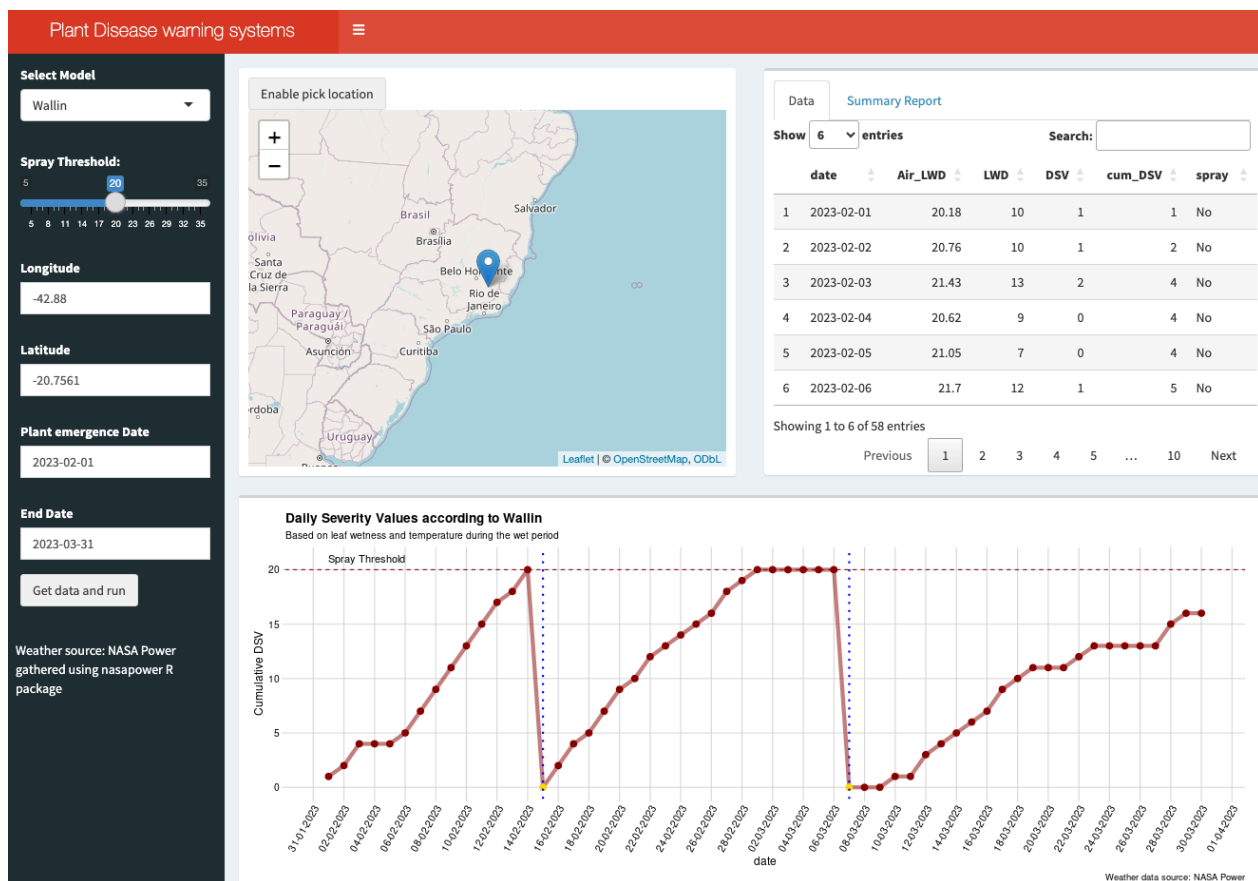


Figure 20.6: Screenshot of a web-based warning system for plant diseases based on Wallin's and FAST rules

Infection risk during defined periods

Many diseases like late blight of potatoes and early blight of tomatoes, as seen in the previous section, require continuous (season long) risk monitoring to indicate multiple fungicide sprays. However, some warning systems assess risk at only one time point, such as those crops that are most vulnerable to diseases during specific growth phases. In these cases, a single warning is issued, such as the risk of disease occurrence or occurrence at an intensity above a threshold (e.g. incidence > 30%).

Some of these systems use statistical (risk algorithms) to predict probabilities of occurrence of the event (yes/no situation) based on weather variables that occur over specific periods of time, usually when primary infections take place. For example, *Fusarium head blight* (FHB, caused by *Fusarium graminearum*) disease depends on weather events that occur around the flowering stage, when the crop is most vulnerable. This is also the case of *Sclerotinia* (white mold) diseases that affect flowers and the presence/absence of apothecia is key for risk assessment (Willbur et al., 2018). Such events of binary nature can be predicted using risk algorithms that consider weather variables, cultivar susceptibility, soil moisture, etc. A commonly used algorithm is the logistic regression model which deals with a binary classification.

Logistic regression: Ohio FHB models

FHB risk models were developed in the United States in the early 2000s to predict the risk of an epidemic, or when disease severity in the field was greater than 10% (De Wolf et al., 2003). In the paper, the epidemic cases were classified as 0 and 1, according to that threshold, and several logistic regression models were fitted to the data using a summary of weather-related

variables from two defined periods: seven days prior to flowering date and 10 days following flowering date (50% of wheat heads with anthers).

Among several models, the authors found that a single variable model (model A) exhibited a good accuracy (0.83). The selected variable consisted of a combination of temperature and humidity for the 10-day period after flowering. Named TRH9010, it corresponds to the total number of hours, within the 10-day post-flowering period, when the temperature (T) was ≥ 15 and ≤ 30 °C and relative humidity (RH) was $>90\%$.

Let's implement this model in R and calculate the probability of infection. As usual, we first need to download hourly data from NASA POWER project for a period of three years. Let's work with data from Wooster, OH, United States.

```
library(nasapower)
weather <- get_power(
  community = "ag",
  lonlat = c(-81.9399, 40.7982),
  pars = c("RH2M", "T2M", "PRECTOTCORR"),
  dates = c("2020-01-01", "2022-12-31"),
  temporal_api = "hourly"
)
head(weather)
```

— NASA/POWER Source Native Resolution Hourly Data —

Dates (month/day/year): 01/01/2020 through 12/31/2022 in LST

Location: Latitude 40.7982 Longitude -81.9399

Elevation from MERRA-2: Average for 0.5 x 0.625 degree lat/lon region = 309.08 meters

The value for missing source data that cannot be computed or is outside of the sources availability range: NA

Parameter(s):

Parameters:

RH2M MERRA-2 Relative Humidity at 2 Meters (%) ; T2M MERRA-2 Temperature at 2 Meters (C) ; PRECTOTCORR MERRA-2 Precipitation Corrected (mm/hour)

A tibble: 6 x 9

	LON	LAT	YEAR	MO	DY	HR	RH2M	T2M	PRECTOTCORR
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	-81.9	40.8	2020	1	1	0	86.5	-3.55	0.01
2	-81.9	40.8	2020	1	1	1	88.9	-3.92	0.01
3	-81.9	40.8	2020	1	1	2	88.3	-3.74	0.01
4	-81.9	40.8	2020	1	1	3	85.4	-3.48	0
5	-81.9	40.8	2020	1	1	4	80.2	-3.18	0
6	-81.9	40.8	2020	1	1	5	75.7	-3.05	0

We can then create a function named `calculate_TRH9010()`. This function calculates the number of hours when specified condition is met (i.e., hours when $15 \leq T \leq 30$ & $RH > 90$), the probability (p) of an epidemic, and the classification of the epidemic (epi) as 0 or 1 based on a defined threshold for p (0.36 in this case) (De Wolf et al., 2003). The function's arguments are the weather data frame, YEAR, MO, and DY, allowing us to estimate the risk for a specific day (the flowering date) within the downloaded period. It's important to note that, in the original model, the variables were normalized (its value divided by the maximum observed value). Thus, the hour count for TRH9010 should be divided by the constant 136.

```
calculate_TRH9010 <- function(data, YEAR, MO, DY){
  start_row <- which(data$YEAR == YEAR & data$MO == MO & data$DY == DY & data$HR == 0)[1]
  if ((nrow(data) - start_row) < 10 * 24){
    return(list(TRH9010 = NA, p = NA, epi = NA))
  }
  subset_data <- data[start_row:(start_row + 10 * 24 - 1),]
  condition_met <- with(subset_data, T2M >= 15 & T2M <= 30 & RH2M > 90)
  total_hours <- sum(condition_met)
  y <- -3.3756 + 6.8128 * (total_hours / 136) # divide by maximum of 136 as in the paper
  p <- exp(y) / (1 + exp(y))
  epi <- ifelse(p > 0.36, 1, 0)
  return(list(TRH9010 = total_hours, p = p, epi = epi))
}
```

Now we apply the function for the 1st of June 2021 as flowering date. The output is the hour count, the probability and the epidemic classification.

```
results <- calculate_TRH9010(weather, YEAR = 2021, MO = 6, DY = 1)
results
```

```
$TRH9010
[1] 70

$p
[1] 0.5327003

$epi
[1] 1
```

We may want to apply the function for a series of days within a given month. Let's write code to predict the risk for the first 15 days of June using a `for()` loop which will store the results in a list, but we can further transform to a dataframe using `map_dfr()` function.

```
num_days <- 15
all_results <- list()
for(day in 1:num_days){
```



```

all_results[[paste0("", day)]] <- calculate_TRH9010(weather, YEAR = 2021, MO = 6, DY = day)
}

# transform results to a dataframe
df <- all_results |>
  map_dfr(~ as.data.frame(t(.)), .id = "day")

# Change variables to numeric
df$p <- as.numeric(df$p)
df$day <- as.numeric(df$day)
df$epi <- as.numeric(df$epi)

```

Finally we plot the results.

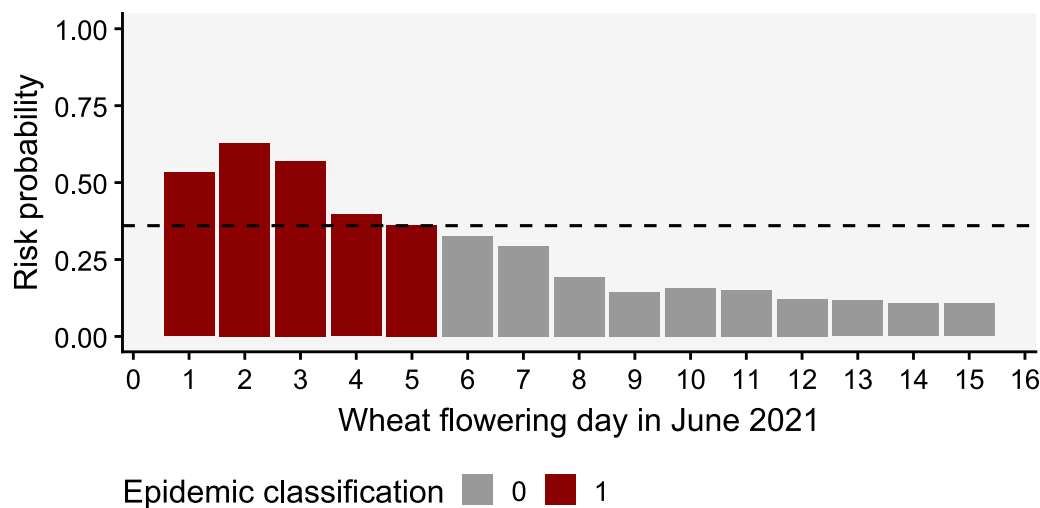
```

ggplot(df, aes(day, p, fill = factor(epi))) +
  geom_col()+
  scale_x_continuous(n.breaks = 15)+
  scale_fill_manual(values = c("grey60", "darkred"))+
  geom_hline(yintercept = 0.36, linetype = 2)+
  r4pde::theme_r4pde(font_size = 12)+
  ylim(0,1)+
  theme(legend.position = "bottom")+
  labs(x = "Wheat flowering day in June 2021", y = "Risk probability",
       fill = "Epidemic classification",
       title = "FHB risk by Ohio (TRH9010) model",
       subtitle = "Location: Wooster, Ohio",
       caption = "Model source: De Wolf et al. (2002)")

```

FHB risk by Ohio (TRH9010) model

Location: Wooster, Ohio



Model source: De Wolf et al. (2002)

Mechanistic models: Brazilian FHB model

Different from the previous approach, where statistical models are fitted to data, some models are called mechanistic, dynamic or process-based. These are designed based on the knowledge of the processes of the disease cycle and the factors that affect these processes (González-Domínguez et al., 2023). These usually operate on an hourly to daily basis and can

be quite complex and sophisticated with numerous examples being found in the literature (Caffi et al., 2007; 2010; Rossi et al., 2014; Salotti et al., 2022; Salotti & Rossi, 2023).

One example is a mechanistic model for FHB developed in Brazil. At its core, the model is based on the daily simulation of the flowering process, the most vulnerable phase of the wheat crop, and events that contribute to infect the flowers. The original model (Del Ponte et al., 2005) integrates several submodels that estimate the daily proportion of flowers, a density of inoculum in a "spore cloud" and a proportion of infected flowers - all these variables being driven by daily variables of temperature and wetness variables such as rainfall and relative humidity.

A modified, simplified version of the model that takes into account only the flowering process and weather variables that drive infection, was written in R and is presented here for illustration.

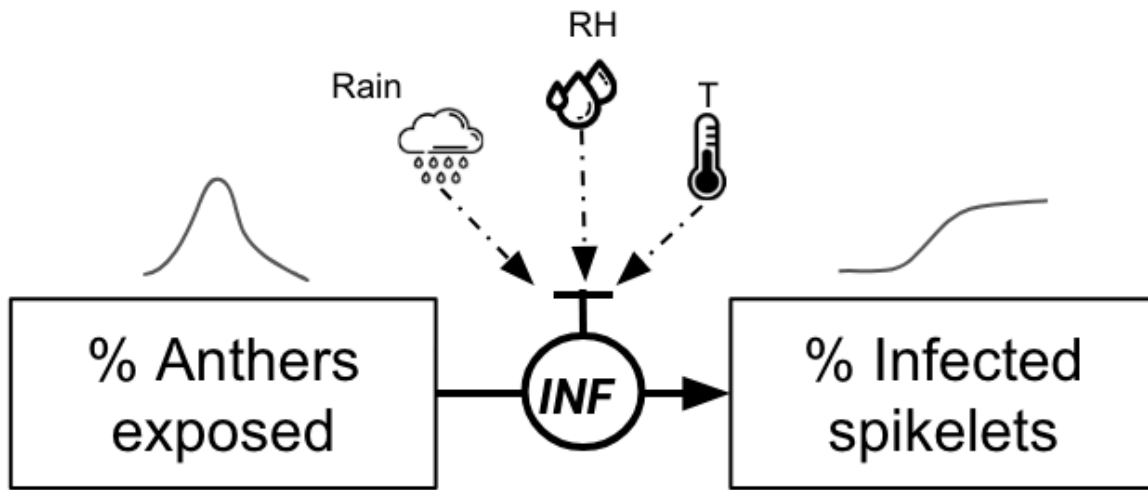


Figure 20.7: A simplified version of a mechanistic Fusarium head blight model that takes into account the amount of flowers on each day which will be infected depending on rules that define a wet period and temperature during the wet period.

The function `simulate_FHB()` requires a starting date and weather data frame (adapted here to use NASA POWER data set) to run a simulation during 20 days.

Among the several simplifications, which were based on the simulated results of the original model, flowering dynamics was assumed to follow a normal distribution over 10 days, and is adjusted to ensure that the peak in the proportion of anthers is set to 0.8. An adjustment is made so that anthers are set to 0 after the 10th day.

$$[\text{antherproportions} = \begin{cases} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) & \text{for } x=1,\dots,10 \\ 0 & \text{for } x=11,\dots,20 \end{cases}],$$

where $\mu = 6$ and $\sigma = 2$.

The probability of the infection of the flowers is calculated as in the paper. It is based on an equation that considers daily temperature (T) and gives the proportion of flowers getting infected on any given day when a **condition for infection** is met.

$$INF = 0.001029 \times \exp(0.1957 \times T)$$

There are three conditions for an infection event, similar to what is described in the article (Del Ponte et al., 2005):

- **Continuous 2-day Rain & High Humidity:** There should be rain (> 0.5 mm) and the average humidity over two consecutive days should be high (> 80%).
- **Rain Followed by Dry Day & High Humidity:** Rain and high humidity (> 80%), followed by a day without rain but still with high humidity (> 85%).
- **Dry Day & High Humidity Followed by Rain:** A day without rain but with high humidity (> 85%), followed by a day with both rain and high humidity (> 80%).

Infection then occurs when any of the above conditions are met. In this simpler version (without the inoculum factor), the proportion of infected spikelets, assuming that infected anthers will give rise to infected spikelets, is given by

$$InfectedSpikelets = AntherProportions \times INF$$

```
simulate_FHB <- function(start_date, weather_data) {
  start_date <- as.Date(start_date, format = "%Y-%m-%d")

  # Extract weather data for 20 days starting from the provided start_date
  weather_subset <- subset(weather_data, YYYYMMDD >= start_date & YYYYMMDD < start_date + 20)

  # Simulation for 20 days
  x <- 1:20

  # Normal distribution formula for 10 days using fixed peak day (mu) and spread (sigma)
  mu <- 6
  sigma <- 2
  anther_proportions <- c((1 / (sigma * sqrt(2 * pi))) * exp(-0.5 * ((x[1:10] - mu) / sigma)^2), rep(0, 10))

  # Adjust the values to have a peak of 0.8
  max_value <- max(anther_proportions)
  scaling_factor <- 0.8 / max_value
  anther_proportions <- anther_proportions * scaling_factor # This scales the entire vector

  # Zero out the anthers after the day they should be zero
  anther_proportions[11:20] <- 0

  # Calculate the INF value
  INF <- 0.001029 * exp(0.1957 * weather_subset$T2M)

  # Calculate average RH for two consecutive days
  avg_RH2M_2days <- (weather_subset$RH2M + c(weather_subset$RH2M[-1], NA)) / 2

  # Daily rain, humidity and their values on the previous and next day
  rain <- weather_subset$PRECTOTCORR
  rh <- weather_subset$RH2M
  rain_prev <- c(NA, head(rain, -1))
  rain_next <- c(tail(rain, -1), NA)
  rh_prev <- c(NA, head(rh, -1))
  rh_next <- c(tail(rh, -1), NA)

  # Rain and high humidity (average of the day and the next day)
  condition1 <- (rain > 0.3) & (avg_RH2M_2days > 80)

  # 1. Continuous 2-day rain and high humidity
```

```

consecutive_rain_high_humidity <- condition1 & c(FALSE, head(condition1, -1))

# 2. Dry day with high humidity followed by a rainy day
preceded_condition <- condition1 & (rh_prev > 85) & (rain_prev <= 0.3)

# 3. Rainy day followed by a dry day with high humidity
succeeded_condition <- condition1 & (rh_next > 85) & (rain_next <= 0.3)

# Combine conditions (missing values at the edges count as FALSE)
infection_trigger <- (consecutive_rain_high_humidity | preceded_condition |
                      succeeded_condition) %in% TRUE

# Calculate infected anthers based on conditions
infected_spikelets <- ifelse(infection_trigger, anther_proportions * INF, 0)

# Cumulative infected anthers
cumulative_infected_spikelets <- cumsum(infected_spikelets)

# Compile results
result_df <- data.frame(
  Date = weather_subset$YYYYMMDD,
  T2M = weather_subset$T2M,
  RAIN = weather_subset$PRECTOTCORR,
  RH2M = weather_subset$RH2M,
  Flowers = round(anther_proportions, 3),
  InfSpikelets = round(infected_spikelets, 3),
  CumInfSpikelets = round(cumulative_infected_spikelets, 3)
)
return(result_df)
}

```

Let's download the data again for the location of Wooster, Ohio during a 9-year period.

```

library(nasapower)
wooster <- get_power(
  community = "ag",
  lonlat = c(-81.9399, 40.7982),
  pars = c("RH2M", "T2M", "PRECTOTCORR"),
  dates = c("2015-01-01", "2023-09-30"),
  temporal_api = "daily"
)
wooster

```

— NASA/POWER Source Native Resolution Daily Data —

Dates (month/day/year): 01/01/2015 through 09/30/2023 in LST

Location: latitude 40.7982 longitude -81.9399

```
elevation from MERRA-2: Average for 0.5 x 0.625 degree lat/lon region = 309.08
meters
```

```
The value for missing source data that cannot be computed or is outside of the
sources availability range: NA
```

```
parameter(s):
```

```
Parameters:
```

```
RH2M MERRA-2 Relative Humidity at 2 Meters (%) ; T2M MERRA-2 Temperature at 2
Meters (C) ; PRECTOTCORR MERRA-2 Precipitation Corrected (mm/day)
```

```
# A tibble: 3,195 x 10
  LON  LAT  YEAR  MM  DD  DOY YYYYMMDD  RH2M  T2M PRECTOTCORR
  <dbl> <dbl> <dbl> <int> <int> <int> <date>    <dbl> <dbl>    <dbl>
1 -81.9 40.8 2015     1     1     1 2015-01-01  67.4 -5.04         0
2 -81.9 40.8 2015     1     2     2 2015-01-02  79.7 -2.1          0
3 -81.9 40.8 2015     1     3     3 2015-01-03  94.0  1.96        25.7
4 -81.9 40.8 2015     1     4     4 2015-01-04  88.8  3.89         8.65
5 -81.9 40.8 2015     1     5     5 2015-01-05  63.1 -9.22         0.54
6 -81.9 40.8 2015     1     6     6 2015-01-06  76.6 -9.91         2.3
7 -81.9 40.8 2015     1     7     7 2015-01-07  70.9 -12.3         0.58
8 -81.9 40.8 2015     1     8     8 2015-01-08  72.2 -13.7         0.54
9 -81.9 40.8 2015     1     9     9 2015-01-09  75.6 -10.8         1.41
10 -81.9 40.8 2015     1    10    10 2015-01-10  78.7 -14.2         0.02
# i 3,185 more rows
```

Now that we have the data, we can simulate FHB for a specific start flowering date.

```
start_date <- "2018-05-15"
result <- simulate_FHB(start_date, wooster)
head(result, 15)
```

	Date	T2M	RAIN	RH2M	Flowers	InfSpikelets	CumInfSpikelets
1	2018-05-15	19.08	4.11	88.20	0.035	0.000	0.000
2	2018-05-16	17.18	0.09	84.75	0.108	0.000	0.000
3	2018-05-17	18.76	0.04	80.52	0.260	0.000	0.000
4	2018-05-18	18.49	0.15	82.75	0.485	0.000	0.000
5	2018-05-19	19.95	9.49	86.50	0.706	0.000	0.000
6	2018-05-20	16.60	1.43	87.37	0.800	0.021	0.021
7	2018-05-21	17.75	15.55	83.50	0.706	0.023	0.045
8	2018-05-22	19.93	9.70	87.09	0.485	0.025	0.069
9	2018-05-23	16.47	0.05	76.00	0.260	0.000	0.069
10	2018-05-24	20.43	0.00	60.82	0.108	0.000	0.069
11	2018-05-25	21.20	0.03	68.93	0.000	0.000	0.069
12	2018-05-26	21.06	9.51	86.16	0.000	0.000	0.069
13	2018-05-27	23.13	0.60	80.81	0.000	0.000	0.069
14	2018-05-28	25.53	0.01	73.79	0.000	0.000	0.069
15	2018-05-29	24.91	0.01	80.71	0.000	0.000	0.069

Finally we can plot the dynamics of the flowering and the percent of infected spikelets during the period of 20 days after first flowers appear in the field.

```
# Plotting the data
library(tidyverse)
p1 <- result |>
  ggplot(aes(x = Date)) +
  geom_area(aes(y = Flowers,
               color = "Proportion of flowers"),
            linewidth = 1, fill = "yellow", alpha = 0.5) +
  geom_line(aes(y = CumInfSpikelets,
               color = "Cum. Infected Spikelets"),
            linewidth = 1) +
  labs(title = "FHB simulation model",
       subtitle = paste("% Infected spikelets = ",
                        round(max(result$CumInfSpikelets*100, na.rm = TRUE), 2), "%"),
       y = "Proportion", x = "Date", color = "Legend") +
  r4pde::theme_r4pde(font_size = 12) +
  theme(legend.position = "bottom")
```

```
p2 <- result |>
  ggplot(aes(x = Date)) +
  geom_col(aes(x = Date, y = RAIN)) +
  geom_line(aes(y = T2M)) +
  geom_line(aes(y = RH2M), linetype = 2) +
  labs(y = "Value",
       title = "Rain, T and RH") +
  r4pde::theme_r4pde(font_size = 12)
```

```
library(patchwork)
p1 / p2
```

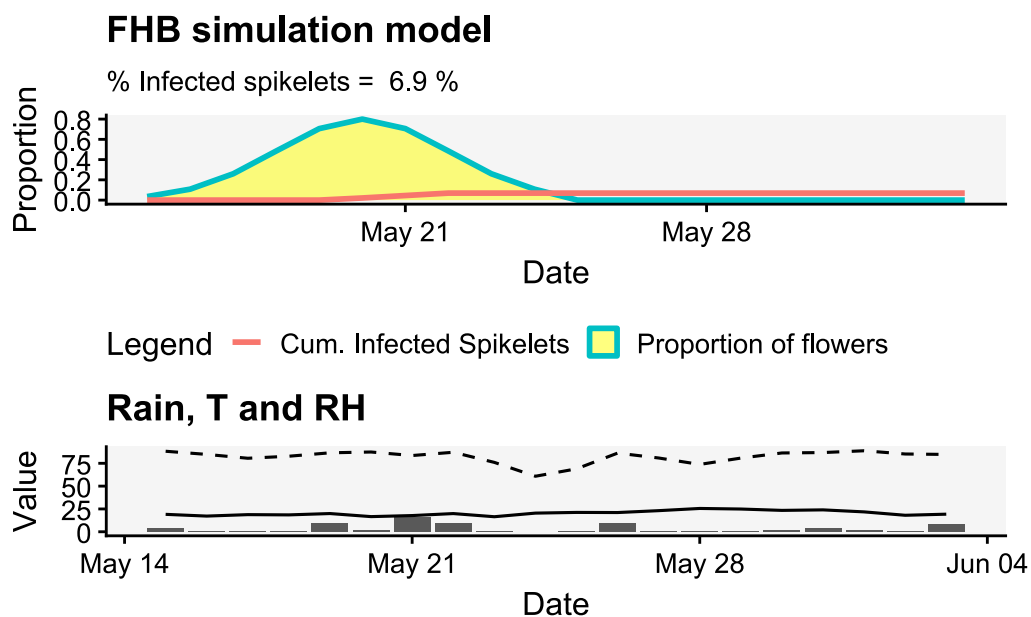


Figure 20.8: Simulated proportion of flowers and cumulative proportion of infected spikelets for a flowering start date of 15 May 2018 in Wooster, Ohio (top), and the daily rain, temperature and relative humidity used in the simulation (bottom)

As in the previous section, we can apply the function for a series of days and see how the percent of infected spikelets fluctuates over time considering each day as a starting day for flowering.

```
# Vector of dates
dates_vector <- seq(as.Date("2019-05-15"), as.Date("2019-05-30"), by="days")

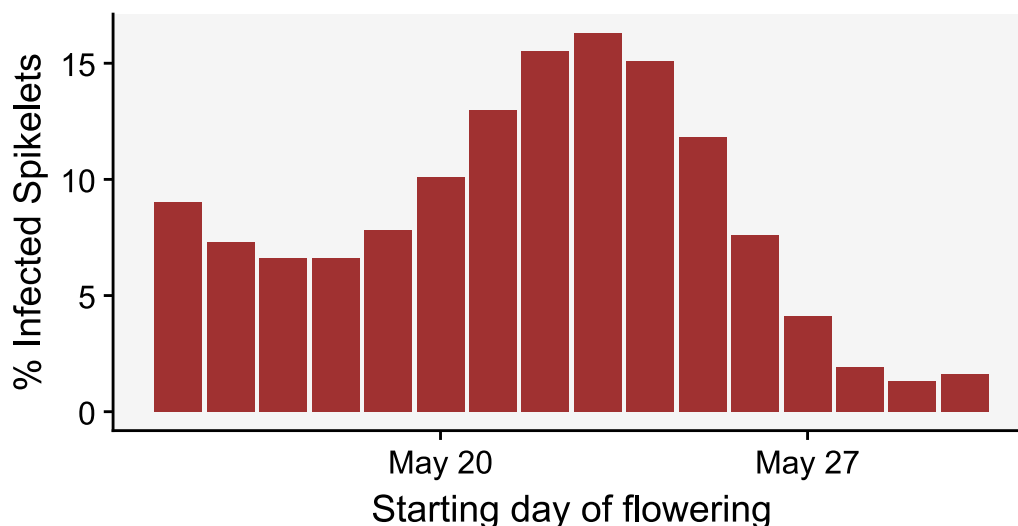
# Apply function and extract max cumulative infection
max_cumulative_infections <- sapply(dates_vector, function(date) {
  result <- simulate_FHB(date, wooster)
  max(result$CumInfSpikelets, na.rm = TRUE)
})

df_results <- data.frame(
  Date = dates_vector,
  MaxCumulativeInfection = max_cumulative_infections
)

# plot the results
df_results |>
  ggplot(aes(Date, MaxCumulativeInfection*100))+
  geom_col(fill = "darkred", alpha = 0.8)+
  labs(title = "FHB simulation model",
       subtitle = "Wooster, Ohio - 2019",
       x = "Starting day of flowering", y = "% Infected Spikelets")+
  r4pde::theme_r4pde(font_size = 14)
```

FHB simulation model

Wooster, Ohio - 2019



Practical considerations

While initial disease warning systems faced underuse, this was attributed to their primary deployment by advisors rather than growers directly, and an evolution where users, through experience, develop simplified rules that diminish the need for constant system consultation. Notable examples, such as the CPO in Denmark and FARMSCAPE in Australia, have

documented this learning process, leading to less reliance on such tools. Growers often favor 'good enough' solutions that evolve from system use, which may reduce the need for direct interaction with the system over time.

However, certain diseases with specific control timing requirements, regulatory pressures, or those affecting high-value crops necessitate direct engagement with warning systems. In these scenarios, growers prioritize yield consistency and economic security over reduced pesticide usage, especially when faced with asymmetric risks associated with incorrect management decisions. It has been suggested that disease warning systems might be more beneficial and accepted for crops of intermediate or lower value, where the cost of mismanagement is more balanced. Essential to the success and adoption of these systems is the active involvement of end-users throughout the development process to ensure the system's features align with their practical farming goals and constraints.

21. Disease modeling

Introduction

As seen in the previous chapter, plant disease modeling is a crucial tool for predicting disease dynamics and informing management decisions when integrated into decision support systems. By leveraging models, researchers and practitioners can anticipate disease outbreaks, assess potential risks, and implement timely interventions to mitigate losses (Rossi et al., 2010; Savary et al., 2018).

Mathematical modeling involves representing empirical phenomena and experimental outcomes using mathematical functions. The data used for these models may be collected specifically for modeling purposes or drawn from existing experiments and observations originally conducted to address different research questions, with such data often found in the literature (Hau & Kranz, 1990).

Mathematical models integrating plant, disease, and environmental - in most cases weather-based variables - factors have been developed since the mid-1900s (see the recent review by González-Domínguez et al. (2023)). Dynamic modeling of disease epidemics gained traction in the early 1960s with foundational work by Vanderplank and Zadoks, setting the stage for future advancements. Since then, researchers have contributed extensively to model development, mainly focusing on the plant disease cycle, which outlines pathogen development stages, such as dormancy, reproduction, dispersal, and pathogenesis, driven by interactions among host, pathogen, and environmental factors (De Wolf & Isard, 2007).

A systematic map by Fedele et al. (2022) identified over 750 papers on plant disease models, primarily aimed at understanding system interactions (n = 680). This map revealed that while most models focus on system understanding, fewer are devoted to tactical management (n = 40), strategic planning (n = 38), or scenario analysis (n = 9).

In terms of model development, we can classify the models into two main groups based on the approach taken (González-Domínguez et al., 2023): Empirical or mechanistic approaches, which differ fundamentally in their basis, complexity and application (Figure 21.1).

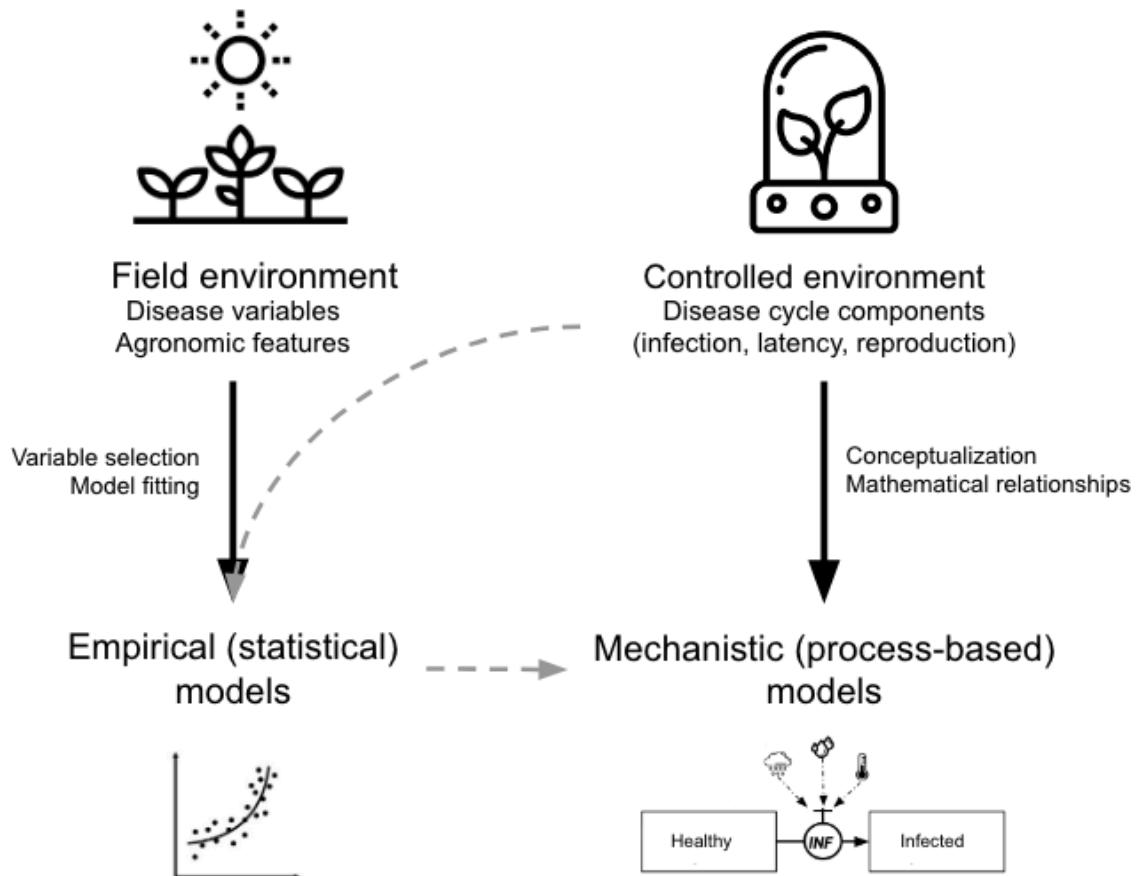


Figure 21.1: Steps of model development from data collection to modeling based on statistical relationships (data-driven) between data collected from field or controlled environment to mechanistic approach based on the elements of the disease cycles (concept-driven).

Empirical models, which emerged in the mid-20th century, rely on data-driven statistical relationships between variables collected under varying field or controlled environments. These models often lack cause-effect understanding, making them less robust and requiring rigorous validation and calibration when applied in diverse environments, especially in regions that did not provide data for model construction. The parameters of the model change every time new data are incorporated during model development.

In contrast, **mechanistic models**, developed from a deep understanding of biological and epidemiological processes, explain disease dynamics based on known system behaviors in response to external variables—a concept-driven approach. These dynamic models quantitatively characterize the state of the pathosystem over time, offering generally more robust predictions by utilizing mathematical equations to describe how epidemics evolve under varying environmental conditions.

Both empirical and mechanistic approaches are valid methodologies extensively used in plant pathology research. The choice between these approaches depends on several factors, including data availability, urgency in model development, and, frequently, the researcher's experience or preference. Empirical models focus on statistical relationships derived directly from data, whereas mechanistic models aim to represent the biological processes of disease progression through linked mathematical equations.

In mechanistic modeling, the equations used to predict specific disease components—such as infection frequency or the latency period—are often empirically derived from controlled experiments. For example, an infection frequency equation is typically based on data collected under specific environmental conditions, with models fitted to accurately describe observed patterns. These process-based models are then built by integrating empirically-derived equations or rules, which collectively simulate the disease cycle. Data and equations are sourced from published studies or generated from new experiments conducted by researchers.

Beyond their practical predictive value, mechanistic models are valuable tools for organizing existing knowledge about a particular disease, helping to identify gaps and guide future research efforts. An example of such work is the extensive collection of comprehensive mechanistic models developed for various plant diseases by the research group led by Prof. Vittorio Rossi in Italy (Rossi et al., 2008; 2014; Salotti et al., 2022; Salotti & Rossi, 2023).

This chapter focuses mainly on empirical modeling. We begin by examining the types of data utilized in model development, focusing on those collected under controlled conditions, such as replicated laboratory or growth chamber experiments, as well as field data collected from several locations and years. We will also analyze real-world case studies, drawing on examples from the literature to replicate and understand model applications. Through these examples, we aim to illustrate the process of fitting models to data and underscore the role of modeling in advancing plant disease management practices.

Controlled environment

In this section, we will demonstrate, using examples from the literature, how statistical models can be fitted to data that represent various stages of the disease cycle.

Research on disease-environment interactions under controlled conditions - such as laboratory or growth chamber studies - lays the groundwork for building foundational models, including infection-based models and sub-models for specific processes like dormancy, dispersal, infection, and latency (De Wolf & Isard, 2007; Krause & Massie, 1975; Magarey et al., 2005).

Growth chambers and phytotrons are essential for testing the effects of individual variables, though these controlled results may not fully replicate field conditions. Laboratory experiments help clarify specific questions by isolating interactions, unlike complex field trials where host, pathogen, and environment factors interact. Polycyclic or “mini epidemic” experiments enable observation of disease dynamics under targeted conditions (Hau & Kranz, 1990; Rotem, 1988).

Once developed, these sub-models can be incorporated into larger mechanistic models that simulate the entire disease cycle, thereby mimicking disease progression over time (Rossi et al., 2008; Salotti & Rossi, 2023). Alternatively, sub-models can also be used in stand-alone predictive systems where the process being modeled - such as infection - is the key factor in determining disease occurrence (MacHardy, 1989; Magarey & Sutton, 2007). For example, infection sub-models can be integrated into prediction systems that help schedule crop protection measures by forecasting when infection risk is highest.

Infection-based models

To model infection potential based on environmental factors, simple rules can be used with daily weather data, such as temperature and rainfall thresholds (Magarey et al., 2002). Simple

decision aids, such as charts and graphs, also exist to help model infection potential by using combinations of daily average temperature and hours of wetness. These tools offer a straightforward approach to evaluate infection risks based on readily available weather data, supporting decision-making without complex modeling (Seem, 1984). However, for many pathogens, hourly data is needed, requiring complex models that track favorable conditions hour by hour. These models start with specific triggers and can reset due to conditions like dryness or low humidity, simulating a biological clock for infection risk assessment (Magarey & Sutton, 2007).

Modeling approaches vary based on available data and model goals. A common method is the matrix approach, like the Wallin potato late blight model, which uses rows for temperature and columns for moisture duration to estimate disease severity (Krause & Massie, 1975) (see previous chapter on warning systems). Bailey enhanced this with an interactive matrix that combines temperature, relative humidity, and duration to assess infection risk across pathogens, making it versatile for various modeling needs (Bailey, 1999).

When infection responses are measured at various temperature and wetness combinations, regression models can be developed to predict infection rates. These models often use polynomial, logistic, or complex three-dimensional response surface equations to represent the relationship between environmental conditions and infection potential. In an excellent review titled *"How to create and deploy infection models for plant pathogens"*, Magarey & Sutton (2007) discusses that many modeling approaches lack biological foundations and are not generic, making them unsuitable for developing a unified set of disease forecast models. While three-dimensional response surfaces, such as those created with sufficient temperature-moisture observations, offer detailed infection responses, they are often too complex and data-intensive for widespread use (see Table 1 adapted from Magarey & Sutton (2007)).

Approach	Strengths	Weaknesses
Matrix (Krause & Massie, 1975; Mills, 1944; Windels et al., 1998)	Easy; converts moisture/temperature combinations into severity values or risk categories. Tried and true approach.	Data to populate matrix may not be readily available.
Regression: • Polynomial (Evans, 1992) • Logistic (Bulger, 1987)	Widely used in plant pathology. Available for many economically important pathogens.	Parameters not biologically based. Requires dataset for development.
Three-dimensional response surface (Duthie, 1997)	Describes infection response in detail.	Parameters not biologically based. Complex, requires extensive data and processing time.
Degree wet hours (Pfender, 2003)	Simple; based on degree hours, commonly used in entomology. Requires only T _{min} and T _{max}	Recently developed; assumes linear thermal response.
Temperature-moisture response function (Magarey et al., 2005)	Simple; based on crop modeling functions, requires only T _{min} , T _{opt} and T _{max}	Recently developed.

In the following sections, we demonstrate how various biologically meaningful models fit infection data, using temperature, wetness duration, or a combination of both as predictors.

Temperature effects

Generalized beta-function

Among several non-linear models that can be fitted to infection responses to temperature, the generalized beta-function is an interesting alternative (Hau & Kranz, 1990). This is a nonlinear model with five parameters. Two of them, namely b and c , have a biological meaning because they are estimates of the minimum and maximum temperature of the biological process under consideration.

We will use a subset of the data obtained from a study conducted under controlled conditions that aimed to assess the influence of temperature on the symptom development of citrus canker in sweet orange (Dalla Pria et al., 2006). The data used here is only for severity on the cultivar Hamlin (plot a in Figure 21.2). The data was extracted using the R package {digitize} as shown [here](#) on [this tweet](#).

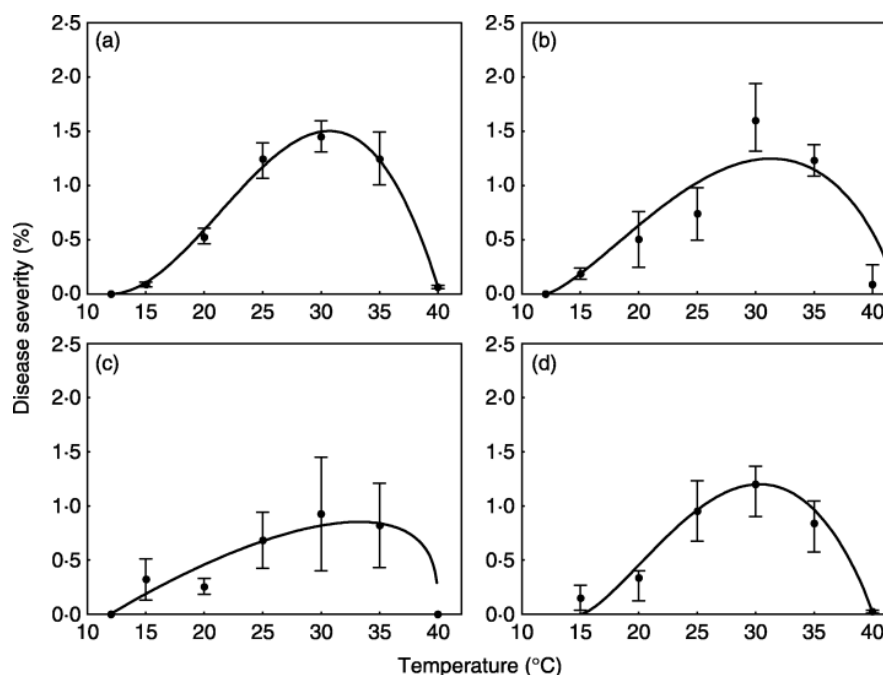


Figure 21.2: Effect of temperature (12, 15, 20, 25, 30, 35 or 40°C) on disease severity of citrus canker on sweet orange cvs Hamlin (a), Natal (b), Pera (c) and Valencia (d) with a leaf wetness duration of 24 h. Each point represents the mean of three repetitions. Vertical bars represent standard errors. Lines show the generalized beta function fitted to data. Source: Dalla Pria et al. (2006)

Let's enter the data manually. Where t is the temperature and y is the severity on leaves.

```
temp <- tibble::tribble(
  ~t, ~y,
  12.0, 0.00,
  15.0, 0.1,
  20.0, 0.5,
  25.0, 1.2,
  30.0, 1.5,
  35.0, 1.2,
  40.0, 0.1
)
```

Fit the generalized beta-function (Hau & Kranz, 1990). The model can be written as:

$$y = a * ((t - b)^d) * ((c - t)^e) \quad (21.1)$$

where b and c represent minimum and maximum temperatures, respectively, for the development of the disease, a , d and e are parameters to be estimated, t is the temperature and y is disease severity. We need the {minpack.lm} library to avoid parameterization issues.

```
library(tidyverse)
library(minpack.lm)
fit_temp <- nlsLM(
  y ~ a * ((t - b) ^ d) * ((c - t) ^ e),
  start = list(
    a = 0,
    b = 10,
    c = 40,
    d = 1.5,
    e = 1
  ),
  algorithm = "port",
  data = temp
)
summary(fit_temp)
```

Formula: $y \sim a * ((t - b)^d) * ((c - t)^e)$

Parameters:

	Estimate	Std. Error	t value	Pr(> t)
a	0.001303	0.006295	0.207	0.855
b	11.999999	4.875412	2.461	0.133
c	40.137236	0.346763	115.748	7.46e-05 ***
d	1.760101	1.193017	1.475	0.278
e	0.830868	0.445213	1.866	0.203

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

Residual standard error: 0.1121 on 2 degrees of freedom

Algorithm "port", convergence message: Relative error between `par` and the solution is at most `ptol`.

```
modelr::rsquare(fit_temp, temp)
```

```
[1] 0.9898275
```

Store the model parameters in objects.

```
fit_temp$m$getAllPars()
```

a	b	c	d	e
0.00130259	11.99999931	40.13723602	1.76010097	0.83086798

```
a <- fit_temp$m$getAllPars()[1]
b <- fit_temp$m$getAllPars()[2]
c <- fit_temp$m$getAllPars()[3]
```

```
d <- fit_temp$m$getAllPars()[4]
e <- fit_temp$m$getAllPars()[5]
```

Create a data frame for predictions at each temperature unit from 10 to 45 degree Celsius.

```
t <- seq(10, 45, 0.1)
y <- a * ((t - b) ^ d) * ((c - t) ^ e)
dat <- data.frame(t, y)
```

Plot the observed and predicted data using {ggplot2} package.

```
library(ggplot2)
library(r4pde)
dat |>
  ggplot(aes(t, y)) +
    geom_line() +
    geom_point(data = temp, aes(t, y)) +
    theme_r4pde(font_size = 16) +
    labs(x = "Temperature", y = "Severity",
         title = "Generalized beta-function")
```

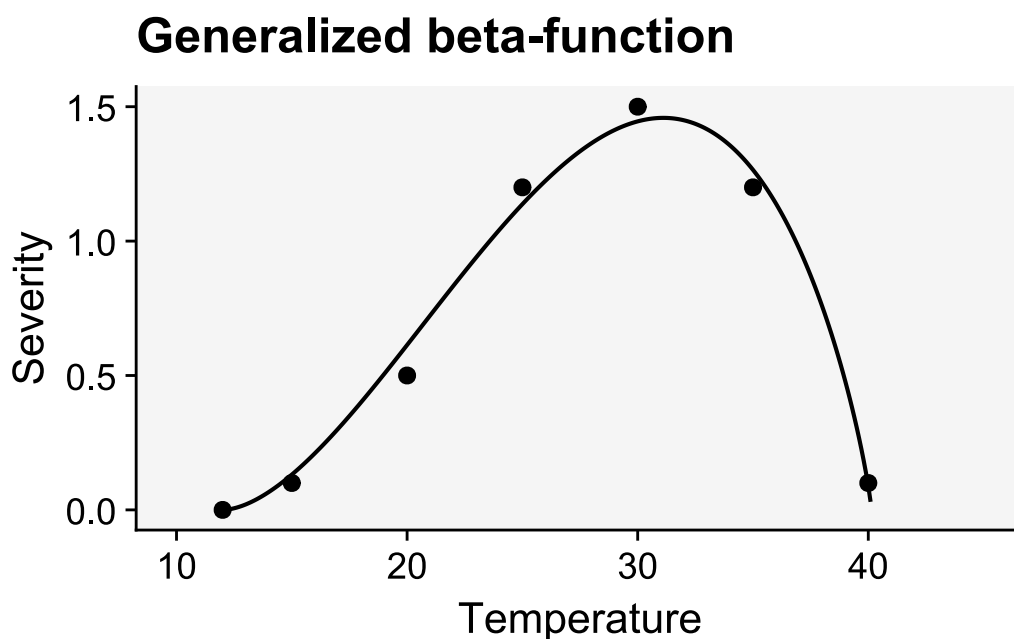


Figure 21.3: Generalized beta function fitted to the severity of citrus canker as a function of temperature (points are the observed values)

Analytis beta function

Ji et al. (2023) tested and compared various mathematical equations to describe the response of mycelial growth to temperature for several fungi associated with Grapevine trunk diseases. The authors found that the beta equation (Analytis, 1977) provided the best fit and, therefore, was considered the most suitable for all fungi.

The model equation for re-scaled severity (0 to 1) as a function of temperature is given by:

$$Y = (a \cdot T_{eq}^b \cdot (1 - T_{eq}))^c \quad ; \quad \text{if } Y > 1, \text{ then } Y = 1$$

where

$$T_{eq} = \frac{T - T_{\min}}{T_{\max} - T_{\min}}$$

T is the temperature in degrees Celsius. $T_{\min}$ is the minimum temperature, $T_{\max}$ is the maximum temperature for severity. The a , b , and c are parameters that define the top, symmetry, and size of the unimodal curve.

Let's rescale (0 to 1) the data on the citrus canker using the function `rescale` of the `{scales}` package.

```
library(scales)
```

```
Attaching package: 'scales'
```

```
The following object is masked from 'package:purrr':
```

```
discard
```

```
The following object is masked from 'package:readr':
```

```
col_factor
```

```
temp$yscaled <- rescale(temp$y)
temp
```

```
# A tibble: 7 × 3
      t     y yscaled
  <dbl> <dbl> <dbl>
1    12     0     0
2    15   0.1  0.0667
3    20   0.5  0.333
4    25   1.2  0.8
5    30   1.5  1
6    35   1.2  0.8
7    40   0.1  0.0667
```

Now we can fit the model using the same `nlsLM` function.

```
# Define the minimum and maximum temperatures
Tmin <- 12
Tmax <- 40

library(minpack.lm)
fit_temp2 <- nlsLM(
  yscaled ~ (a * ((t - Tmin) / (Tmax - Tmin))^b * (1 - ((t - Tmin) / (Tmax - Tmin))))^c,
  data = temp,
  start = list(a = 1, b = 2, c = 3), # Initial guesses for parameters
  algorithm = "port"
)

summary(fit_temp2)
```

```
Formula: yscaled ~ (a * ((t - Tmin)/(Tmax - Tmin))^b * (1 - ((t - Tmin)/(Tmax -
```



```

Tmin))))^c

Parameters:
  Estimate Std. Error t value Pr(>|t|)
a    6.7625    0.3218  21.013 3.03e-05 ***
b    1.9648    0.1030  19.072 4.45e-05 ***
c    1.1607    0.1507   7.701 0.00153 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.03955 on 4 degrees of freedom

Algorithm "port", convergence message: Relative error in the sum of squares is at most `ftol'.

```

```
modelr::rsquare(fit_temp2, temp)
```

```
[1] 0.9948325
```

Let's store the model parameters in objects.

```
fit_temp2$m$getAllPars()
```

```

      a      b      c
6.762509 1.964817 1.160702

```

```

a <- fit_temp2$m$getAllPars()[1]
b <- fit_temp2$m$getAllPars()[2]
c <- fit_temp2$m$getAllPars()[3]

```

Again, we create a data frame for predictions at each temperature unit from 10 to 45 degree Celsius.

```

Tmin <- 12
Tmax <- 40
t <- seq(10, 45, 0.1)
y <- (a * ((t - Tmin) / (Tmax - Tmin))^b * (1 - ((t - Tmin) / (Tmax - Tmin))))^c
dat2 <- data.frame(t, y)

```

And now we can plot the observed and predicted data using {ggplot2} package.

```

library(ggplot2)
library(r4pde)
dat2 |>
  ggplot(aes(t, y)) +
  geom_line() +
  geom_point(data = temp, aes(t, yscaled)) +
  theme_r4pde(font_size = 16) +
  labs(x = "Temperature", y = "Scaled severity",
       title = "Analytis beta function")

```

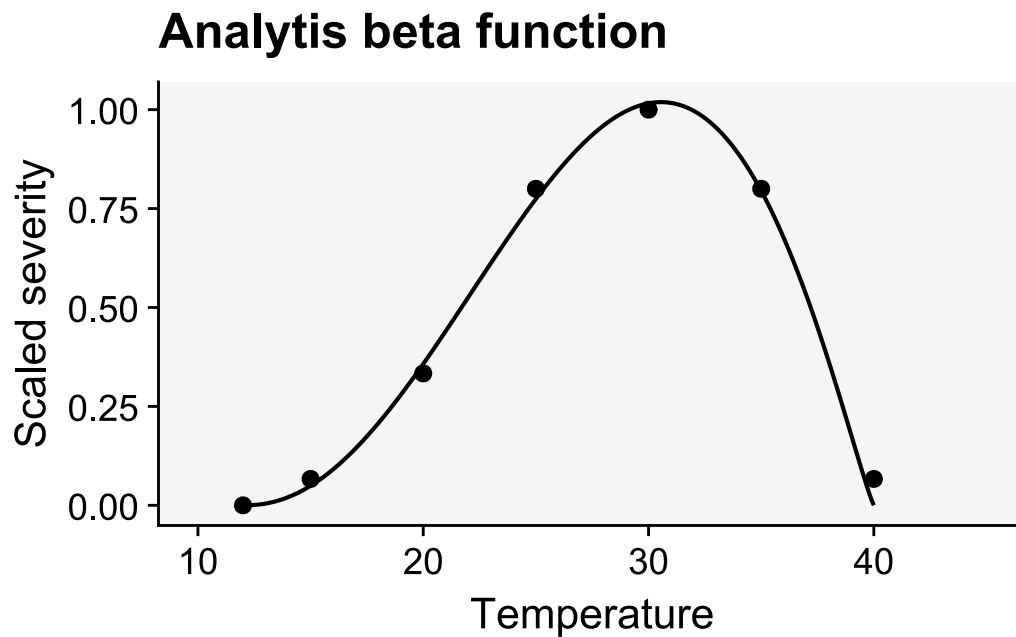


Figure 21.4: Analytis beta function fitted to the rescaled severity of citrus canker as a function of temperature

Moisture effects

Monomolecular function

For this example, we will use a subset of the data obtained from a study conducted under controlled conditions that aimed to assess the effects of moisture duration on the symptom development of citrus canker in sweet orange (Dalla Pria et al., 2006). As in the previous example for temperature effects, the data used here is only for severity on the cultivar Hamlin (plot a in Figure 21.5). The data was also extracted using the R package digitize.

Let's look at the original data and the predictions by the model fitted in the paper.

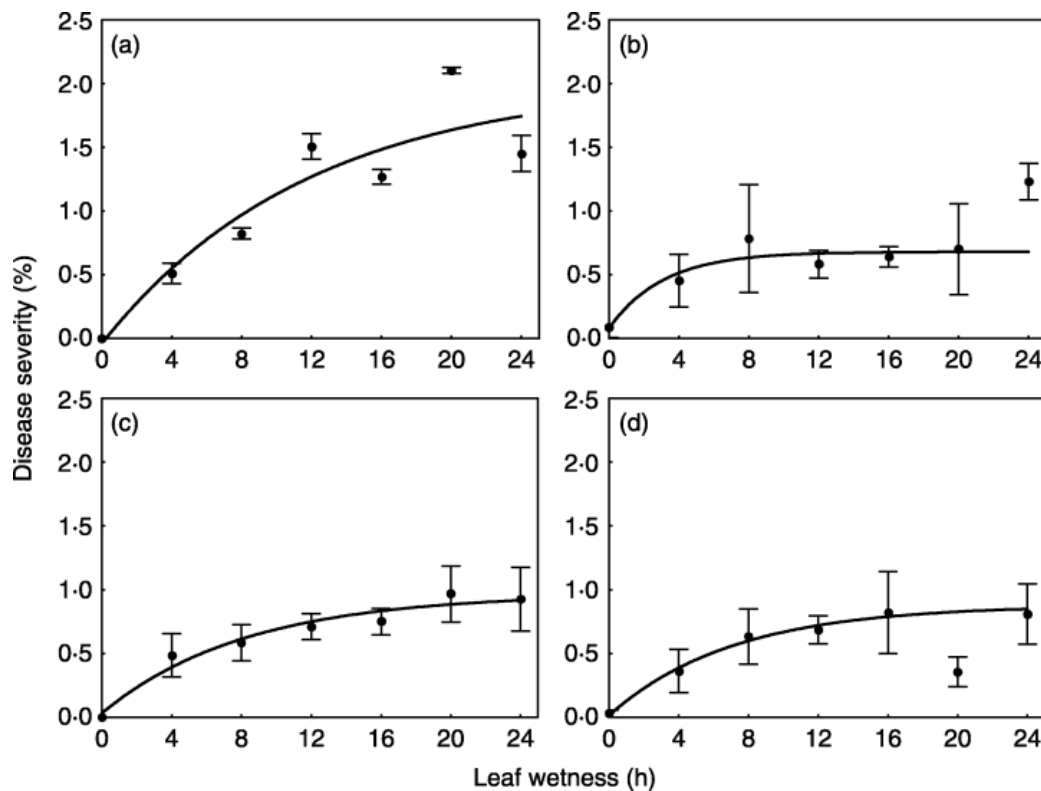


Figure 21.5: Effect of leaf wetness duration (0, 4, 8, 12, 16, 20 or 24 h) on disease severity of citrus canker on sweet orange cvs Hamlin (a), Natal (b), Pera (c) and Valencia (d) at 30°C. Each point represents the mean of three repetitions. Vertical bars represent standard errors. Lines show the monomolecular model fitted to data.

Source: Dalla Pria et al. (2006)

For this pattern in the data, we will fit a three-parameter asymptotic regression model. These models describe a limited growth, where y approaches a horizontal asymptote as x tends to infinity. This equation is also known as Monomolecular Growth, Mitscherlich law or von Bertalanffy law. See [this tutorial](#) for comprehensive information about fitting several non-linear regression models in R.

Again, we enter the data manually. Here x is wetness duration in hours and y is severity.

```
wet <- tibble::tribble(~ x, ~ y,
  0, 0,
  4, 0.50,
  8, 0.81,
  12, 1.50,
  16, 1.26,
  20, 2.10,
  24, 1.45)
```

The model can be written as:

$$y = c1 + (d1 - c1) * (1 - \exp(-x/e1))$$

where c is the lower limit (at $x = 0$), the parameter d is the upper limit and the parameter e (greater than 0) is determining the steepness of the increase as x .

We will solve the model again using the `nlsLM` function. We should provide initial values for the three parameters.

```
fit_wet <- nlsLM(y ~ c1 + (d1 - c1) * (1 - exp(-x / e1)),
               start = list(c1 = 0.5,
                           d1 = 3,
                           e1 = 1),
               data = wet)

summary(fit_wet)
```

```
Formula: y ~ c1 + (d1 - c1) * (1 - exp(-x/e1))
```

```
Parameters:
```

	Estimate	Std. Error	t value	Pr(> t)
c1	-0.04898	0.31182	-0.157	0.8828
d1	2.00746	0.70594	2.844	0.0467 *
e1	11.63694	9.33183	1.247	0.2804

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

```
Residual standard error: 0.3296 on 4 degrees of freedom
```

```
Number of iterations to convergence: 7
```

```
Achieved convergence tolerance: 1.49e-08
```

```
modelr::rsquare(fit_wet, wet)
```

```
[1] 0.8532282
```

Store the value of the parameters in the respective object.

```
HW <- seq(0, 24, 0.1)
c1 <- fit_wet$m$getAllPars()[1]
d1 <- fit_wet$m$getAllPars()[2]
e1 <- fit_wet$m$getAllPars()[3]
y <- (c1 + (d1 - c1) * (1 - exp(-HW / e1)))
dat2 <- data.frame(HW, y)
```

Now we can plot the predictions and the original data.

```
dat2 |>
  ggplot(aes(HW, y)) +
  geom_line() +
  geom_point(data = wet, aes(x, y)) +
  theme_r4pde(font_size = 16) +
  labs(x = "Wetness duration", y = "Severity")
```

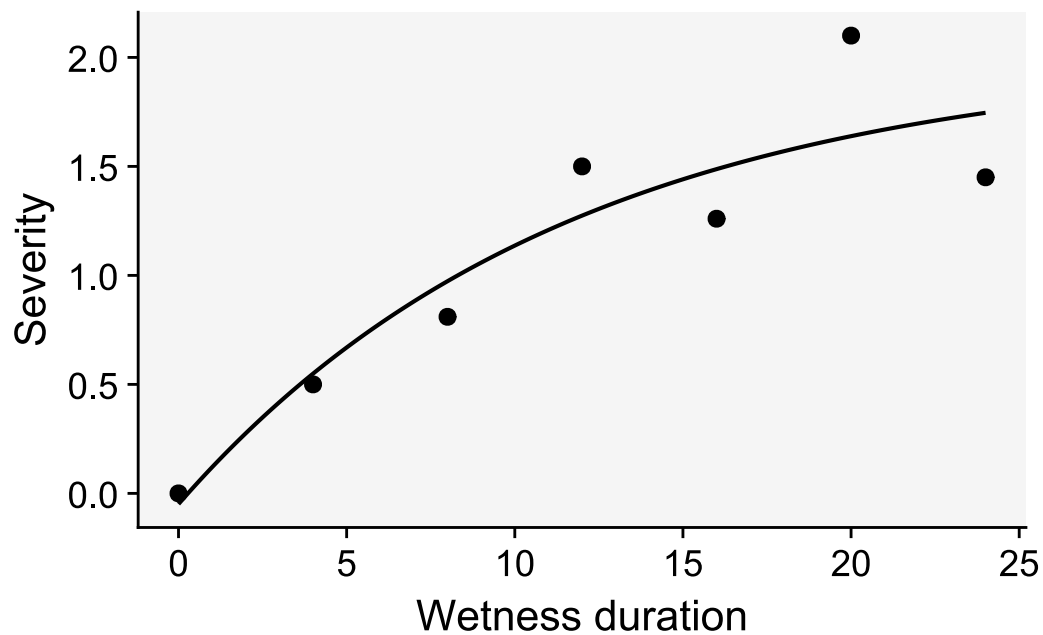


Figure 21.6: Monomolecular model fitted to the severity of citrus canker as a function of leaf wetness duration

Weibull function

In the studies by Ji et al. (2021) and Ji et al. (2023), a Weibull model was fitted to the re-scaled data (0 to 1) on the effect of moisture duration on spore germination or infection. Let's keep working with the re-scaled data on the citrus canker.

The model is given by:

$$y = 1 - \exp(-(a \cdot x)^b)$$

where y is the response variable, x is the moisture duration, a is the scale parameter influencing the rate of infection and b is the shape parameter affecting the curve's shape and acceleration

```
wet$yscaled <- rescale(wet$y)
wet
```

```
# A tibble: 7 × 3
   x     y yscaled
<dbl> <dbl> <dbl>
1     0     0     0
2     4  0.5  0.238
3     8  0.81  0.386
4    12  1.5  0.714
5    16  1.26  0.6
6    20  2.1    1
7    24  1.45  0.690
```

```
fit_wet2 <- nlsLM(
  yscaled ~ 1 - exp(-(a * x)^b),
  data = wet,
  start = list(a = 1, b = 2), # Initial guesses for parameters a and b
```

```
)
summary(fit_wet2)
```

```
Formula: yscaled ~ 1 - exp(-(a * x)^b)

Parameters:
      Estimate Std. Error t value Pr(>|t|)
a  0.07684    0.01296    5.93 0.00195 **
b  1.07610    0.37103    2.90 0.03378 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1404 on 5 degrees of freedom

Number of iterations to convergence: 26
Achieved convergence tolerance: 1.49e-08
```

```
modelr::rsquare(fit_wet2, wet)
```

```
[1] 0.8534077
```

Set the value of the parameters in the respective objects

```
x <- seq(0, 24, 0.1)
a <- fit_wet2$m$getAllPars()[1]
b <- fit_wet2$m$getAllPars()[2]
y <- 1 - exp(-(a * x)^b)
dat3 <- data.frame(x, y)
```

```
dat3 |>
  ggplot(aes(x, y)) +
  geom_line() +
  geom_point(data = wet, aes(x, yscaled)) +
  theme_r4pde(font_size = 16) +
  labs(x = "Wetness duration", y = "Scaled severity")
```

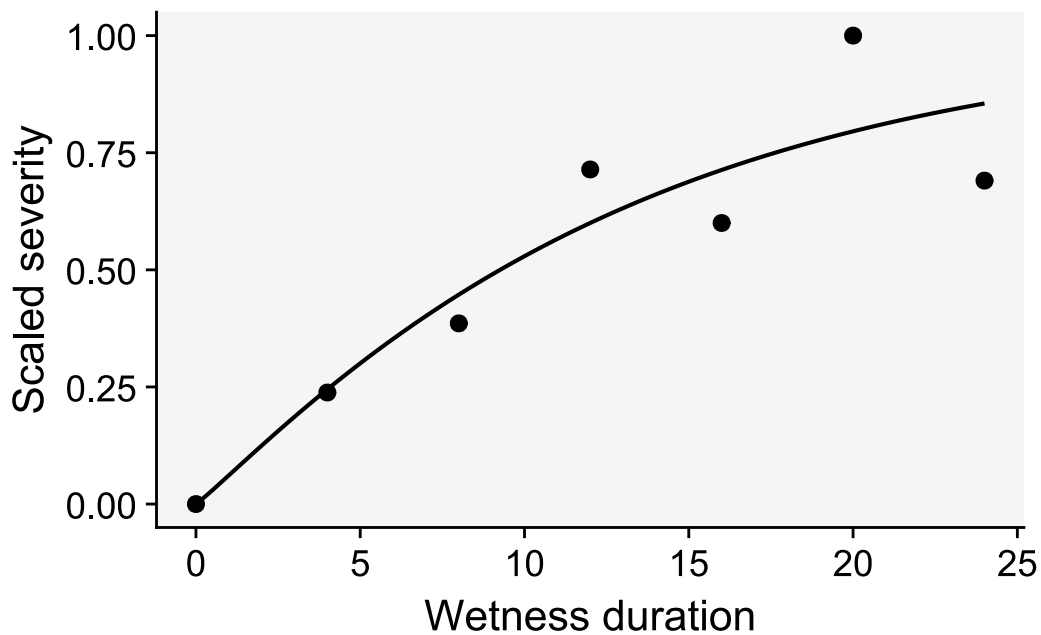


Figure 21.7: Weibull model fitted to the rescaled severity of citrus canker as a function of wetness duration

Integrating temperature and wetness effects

The equations developed for the separate effects can be integrated to create a surface response curve or a simple contour plot. Let's first integrate the generalized beta and the monomolecular models for the original severity data for the citrus canker experiment.

First, we need a data frame for the interaction between temperature t and hours of wetness hw . Then, we obtain the disease value for each combination of t and hw .

```
t <- rep(1:40, 40)
hw <- rep(1:40, each = 40)

# let's fit the two models again and store the parameters in objects
# Temperature effects
fit_temp <- nlsLM(
  y ~ a * ((t - b) ^ d) * ((c - t) ^ e),
  start = list(
    a = 0,
    b = 10,
    c = 40,
    d = 1.5,
    e = 1
  ),
  algorithm = "port",
  data = temp
)
fit_temp$m$getAllPars()
```

a	b	c	d	e
0.00130259	11.99999931	40.13723602	1.76010097	0.83086798

```

a <- fit_temp$m$getAllPars()[1]
b <- fit_temp$m$getAllPars()[2]
c <- fit_temp$m$getAllPars()[3]
d <- fit_temp$m$getAllPars()[4]
e <- fit_temp$m$getAllPars()[5]

## Moist duration effects
fit_wet <- nlsLM(y ~ c1 + (d1 - c1) * (1 - exp(-x / e1)),
               start = list(c1 = 0.5,
                           d1 = 3,
                           e1 = 1),
               data = wet)

c1 <- fit_wet$m$getAllPars()[1]
d1 <- fit_wet$m$getAllPars()[2]
e1 <- fit_wet$m$getAllPars()[3]

dis <-
  (a * (t - b) ^ d) * ((c - t) ^ e) * (c1 + (d1 - c1) * (1 - exp(- hw / e1)))
validation <- data.frame(t, hw, dis)

```

Now the contour plot can be visualized using {ggplot2} and {geomtextpath} packages.

```

library(geomtextpath)
ggplot(validation, aes(t, hw, z = dis)) +
  geom_contour_filled(bins = 8, alpha = 0.7) +
  geom_textcontour(bins = 8,
                  size = 2.5,
                  padding = unit(0.05, "in")) +
  theme_light(base_size = 10) +
  theme(legend.position = "right") +
  ylim(0, 40) +
  labs(y = "Wetness duration (hours)",
       fill = "Severity",
       x = "Temperature (Celsius)",
       title = "Integrating generalized beta and monomolecular")

```

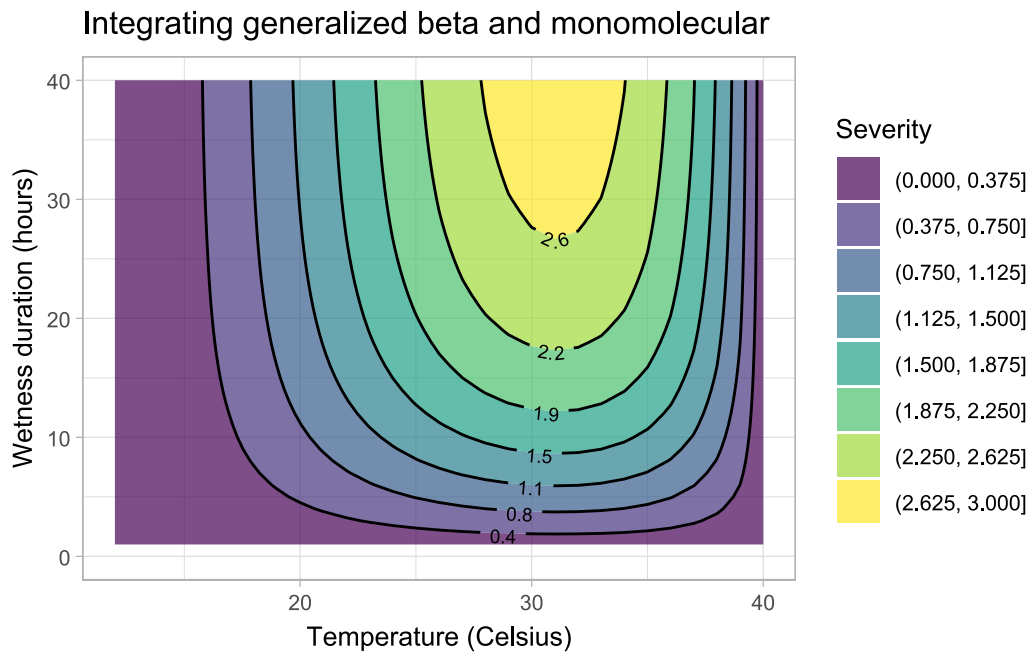



Figure 21.8: Predicted severity of citrus canker as a function of temperature and wetness duration, integrating the generalized beta and monomolecular models

In the second example, let's integrate the Analytis beta and the Weibull model:

```
fit_temp2 <- nlsLM(
  yscaled ~ (a * ((t - Tmin) / (Tmax - Tmin))^b * (1 - ((t - Tmin) / (Tmax - Tmin))))^c,
  data = temp,
  start = list(a = 1, b = 2, c = 3), # Initial guesses for parameters
  algorithm = "port"
)

fit_temp2$m$getAllPars()
```

```
      a      b      c
6.762509 1.964817 1.160702
```

```
a2 <- fit_temp2$m$getAllPars()[1]
b2 <- fit_temp2$m$getAllPars()[2]
c2 <- fit_temp2$m$getAllPars()[3]
```

```
fit_wet2 <- nlsLM(
  yscaled ~ 1 - exp(-(d * x)^e),
  data = wet,
  start = list(d = 1, e = 2), # Initial guesses for parameters a and b
)
```

```
d2 <- fit_wet2$m$getAllPars()[1]
e2 <- fit_wet2$m$getAllPars()[2]
```

```
Tmin <- 12
Tmax <- 40
```

```
dis2 <- (a2 * ((t - Tmin) / (Tmax - Tmin))^b2 * (1 - ((t - Tmin) / (Tmax - Tmin))))^c2 * (1 - exp(-(d2 * x)^e2))
```

```
hw)^e2))

t <- rep(1:40, 40)
hw <- rep(1:40, each = 40)
validation2 <- data.frame(t, hw, dis2)

validation2 <- validation2 |>
  filter(!is.nan(dis2)) |>
  mutate(dis2 = case_when(dis2 < 0 ~ 0,
                          TRUE ~ dis2))
```

Now the plot.

```
ggplot(validation2, aes(t, hw, z = dis2)) +
  geom_contour_filled(bins = 7, alpha = 0.7) +
  geom_textcontour(bins = 7,
                  size = 2.5,
                  padding = unit(0.05, "in")) +
  theme_light(base_size = 10) +
  theme(legend.position = "right") +
  ylim(0, 40) +
  labs(y = "Wetness duration (hours)",
       fill = "Severity",
       x = "Temperature (Celsius)",
       title = "Integrating Analytis beta and Weibull")
```

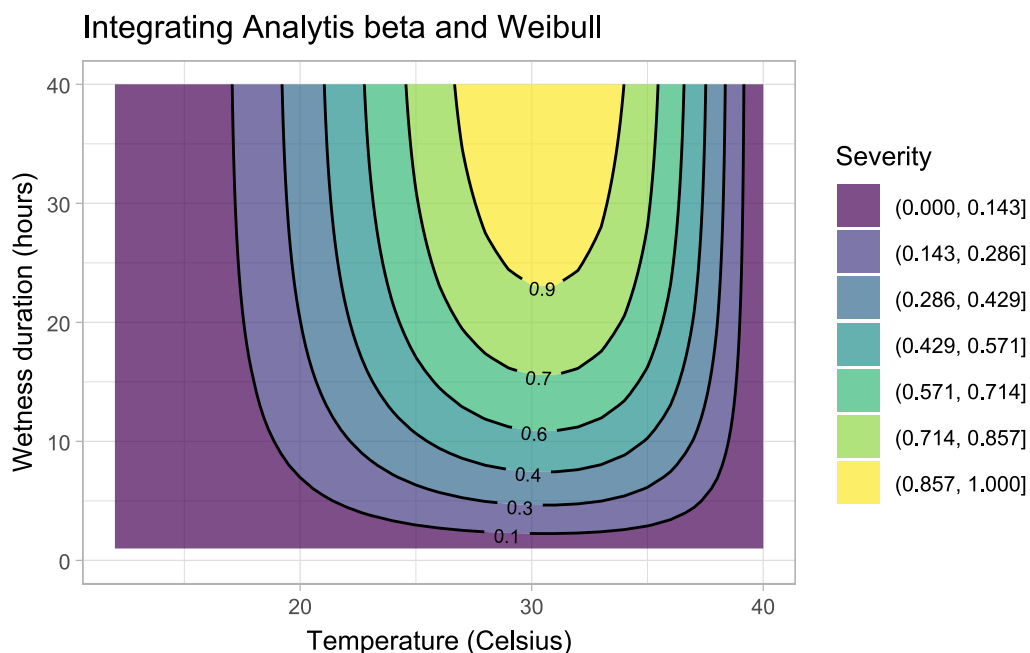


Figure 21.9: Predicted severity as a function of temperature and wetness duration, integrating the Analytis beta and Weibull models

We can create a 3D surface plot to visualize the predictions, as it was used in the original paper. Note that in `plot_ly`, a 3D surface plot requires a matrix or grid format for the `z` values, with corresponding vectors for `x` and `y` values that define the axes. If the data frame (`validation2`) has three columns (`t`, `hw`, and `dis2`), we'll need to convert `dis2` into a matrix format that `plot_ly` can interpret for a surface plot.

i Interactive content

The 3D surface plot is interactive and only available in the online edition of the book:
<https://r4pde.net/prediction-disease-modeling.html>.

Magarey's generic infection model

In the early 2000s, Magarey and collaborators (Magarey et al., 2005) proposed a generic infection model for foliar fungal pathogens, designed to predict infection periods based on limited data on temperature and wetness requirements. The model uses cardinal temperatures (minimum, optimum, maximum) and the minimum wetness duration ($W_{\min}$) necessary for infection. The model can incorporate inputs based on estimated cardinal temperatures and surface wetness duration. These values are available for numerous pathogens and can be consulted in the literature (See table 2 of the paper by Magarey et al. (2005)).

The model utilizes a temperature response function, which is adjusted to the pathogen's minimum and optimum wetness duration needs, allowing it to be broadly applicable even with limited data on specific pathogens. The model was validated with data from 53 studies, showing good accuracy and adaptability, even for pathogens lacking comprehensive data (Magarey et al., 2005).

The function is given by

$$f(T) = \left(\frac{T_{\max} - T}{T_{\max} - T_{\text{opt}}} \right) \times \left(\frac{T - T_{\min}}{T_{\text{opt}} - T_{\min}} \right)^{\frac{T_{\text{opt}} - T_{\min}}{T_{\max} - T_{\text{opt}}}}$$

where T is the temperature, $T_{\min}$ is the minimum temperature, T_{opt} is the optimum temperature, and $T_{\max}$ is the maximum temperature for infection. This is the beta function of Yin et al. (1995) (their Eq. 8b with $\beta = 1$), scaled so that $f(T) = 1$ at the optimum temperature. It is zero at $T_{\min}$ and $T_{\max}$ and reaches its maximum exactly at T_{opt} .

The wetness duration requirement is given by

$$W(T) = \frac{W_{\min}}{f(T)} \leq W_{\max}$$

where $W_{\min}$ is the minimum wetness duration requirement, and $W_{\max}$ is an optional upper limit on $W(T)$.

Let's write the functions for estimating the required wetness duration at each temperature.

```
temp_response <- function(T, Tmin, Topt, Tmax) {
  if (T < Tmin || T > Tmax) {
    return(0)
  } else {
    ((Tmax - T) / (Tmax - Topt)) *
    ((T - Tmin) / (Topt - Tmin)) ^ ((Topt - Tmin) / (Tmax - Topt))
  }
}

# Define the function to calculate wetness duration requirement W(T)
wetness_duration <- function(T, Wmin, Tmin, Topt, Tmax, Wmax = Inf) {
  f_T <- temp_response(T, Tmin, Topt, Tmax)
  if (f_T == 0) {
    return(0) # Infinite duration required if outside temperature range
  }
  W <- Wmin / f_T
  return(min(W, Wmax)) # Apply Wmax as an upper limit if specified
}
```

Let's set the parameters for the fungus *Venturia inaequalis*, the cause of apple scab.

```
# Parameters for Venturia inaequalis (apple scab)
T <- seq(0, 35, by = 0.5)
Wmin <- 6
Tmin <- 1
Topt <- 20
Tmax <- 35
Wmax <- 40.5

# Calculate wetness duration required at each temperature
W_T <- sapply(T, wetness_duration, Wmin, Tmin, Topt, Tmax, Wmax)

temperature_data_applescab <- data.frame(
  Temperature = T,
  Wetness_Duration = W_T
)
```

And now the parameters for the fungus *Phakopsora pachyrhizi*, the cause of soybean rust in soybean.

```
# Parameters for Phakopsora pachyrhizi
T <- seq(0, 35, by = 0.5)
Wmin <- 8
Tmin <- 10
Topt <- 23
Tmax <- 28
Wmax <- 12

# Calculate wetness duration required at each temperature
W_T <- sapply(T, wetness_duration, Wmin, Tmin, Topt, Tmax, Wmax)

temperature_data_soyrust <- data.frame(
  Temperature = T,
  Wetness_Duration = W_T
)
```

We can produce the plots for each pathogen.

```
applescab <- ggplot(temperature_data_applescab, aes(x = Temperature, y = Wetness_Duration)) +
  geom_line(color = "black", linewidth = 1, linetype = 1) +
  theme_r4pde(font_size = 14)+
  labs(x = "Temperature (°C)", y = "Wetness Duration (hours)",
       subtitle = "Venturia inaequalis")+
  theme(plot.subtitle = element_text(face = "italic"))

soyrustr <- ggplot(temperature_data_soyrust, aes(x = Temperature, y = Wetness_Duration)) +
  geom_line(color = "black", linewidth = 1, linetype = 1) +
  theme_r4pde(font_size = 14)+
  labs(x = "Temperature (°C)", y = "Wetness Duration (hours)",
       subtitle = "Phakopsora pachyrhizi")+
  theme(plot.subtitle = element_text(face = "italic"))

library(patchwork)
```

Warning: package 'patchwork' was built under R version 4.5.3

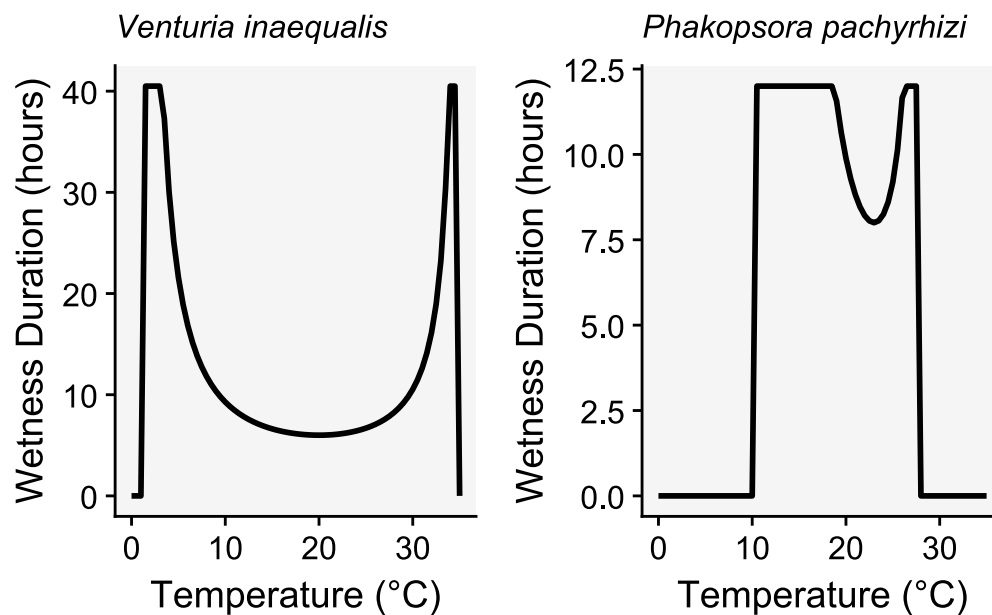


Figure 21.10: Wetness duration required for infection as a function of temperature, estimated with the generic infection model of Magarey et al. (2005) for *Venturia inaequalis* (left) and *Phakopsora pachyrhizi* (right)

Latent period models

The latent period can be defined as “the length of time between the start of the infection process by a unit of inoculum and the start of production of infectious units” (Madden et al., 2007a). The latent period, analogous to the reproductive maturity age of nonparasitic organisms, defines the generation time between infections and is a key factor in pathogen development and epidemic progress in plant disease epidemiology (Vanderplank, 1963). As a critical trait of aggressiveness, especially in polycyclic diseases, it largely determines the potential number of infection cycles within a season, impacting the overall epidemic intensity (Lannou, 2012).

Parabolic function

The effects of temperature on the length of the incubation and latent periods of hawthorn powdery mildew, caused by *Podosphaera clandestina*, were studied by Xu & Robinson (2000). In that work, the authors inoculated the leaves and, each day after inoculation, the upper surface of each leaf was examined for mildew colonies and conidiophores using a pen-microscope ($\times 50$). Sporulation was recorded at the leaf level, noting the number of colonies and the first appearance dates of colonies and sporulation for each leaf.

The latent period (LP) was defined as the time from inoculation to the first day of observed sporulation on the leaf. Due to the skewed distribution of LP across temperatures and inoculations, medians were used to summarize LP rather than means (Xu & Robinson, 2000).

Let’s look at two plots extracted from the paper. The first, on the left-hand side, is the original number of days of the latent period of each evaluated temperature (note: the solid symbol is for constant temperature while the open circle is for fluctuating temperature). On the right-hand side, the relationship between temperature and rates of development of powdery mildew

under constant temperature during the latent periods; the solid line indicates the fitted model. The rate of fungal development was calculated as the reciprocal of the corresponding observed incubation (in hours) and latent periods.

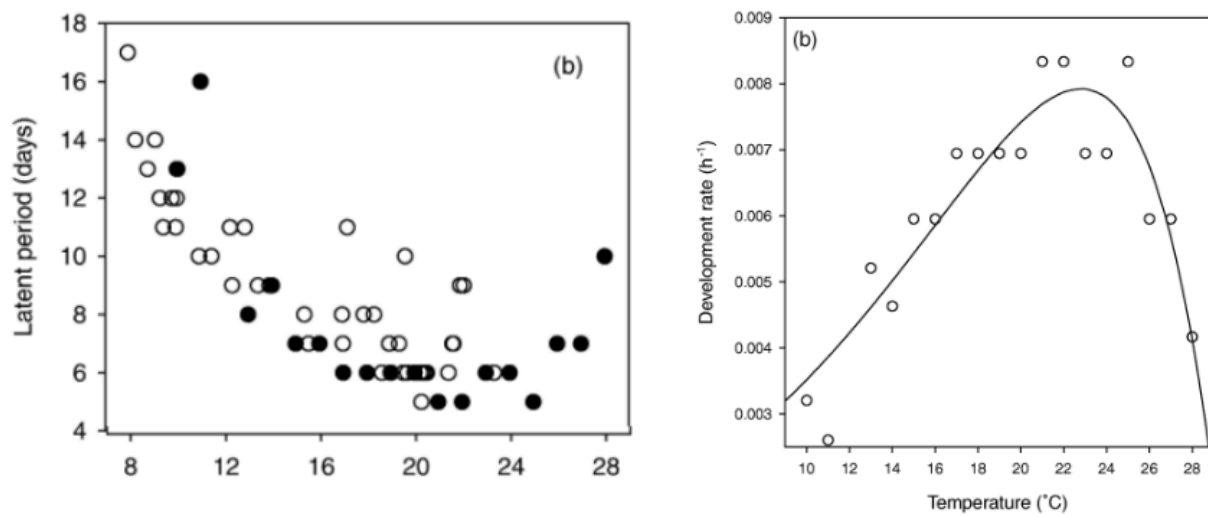


Figure 21.11: Source: Xu & Robinson (2000)

The latent period data in days for the solid black circle (constant temperature) above was extracted using the {digitize} R package.

```
latent <- tibble::tribble(
  ~T, ~days,
  10L, 13L,
  11L, 16L,
  13L, 8L,
  14L, 9L,
  15L, 7L,
  16L, 7L,
  17L, 6L,
  18L, 6L,
  19L, 6L,
  20L, 6L,
  21L, 5L,
  22L, 5L,
  23L, 6L,
  24L, 6L,
  25L, 5L,
  26L, 7L,
  27L, 7L,
  28L, 10L
)
```

Let's reproduce the two plots using the datapoints.

```
#|fig-width: 10
#|fig-height: 4
library(ggplot2)
library(r4pde)
```

```

p_latent <- latent |>
  ggplot(aes(T, days))+
  geom_point()+
  theme_r4pde()

latent_rate <- data.frame(
  T = latent$T, # Scale temperature
  R = 1/latent$days/24
)

p_latent_rate <- latent_rate |>
  ggplot(aes(T, R))+
  geom_point()+
  theme_r4pde()

library(patchwork)
p_latent | p_latent_rate

```

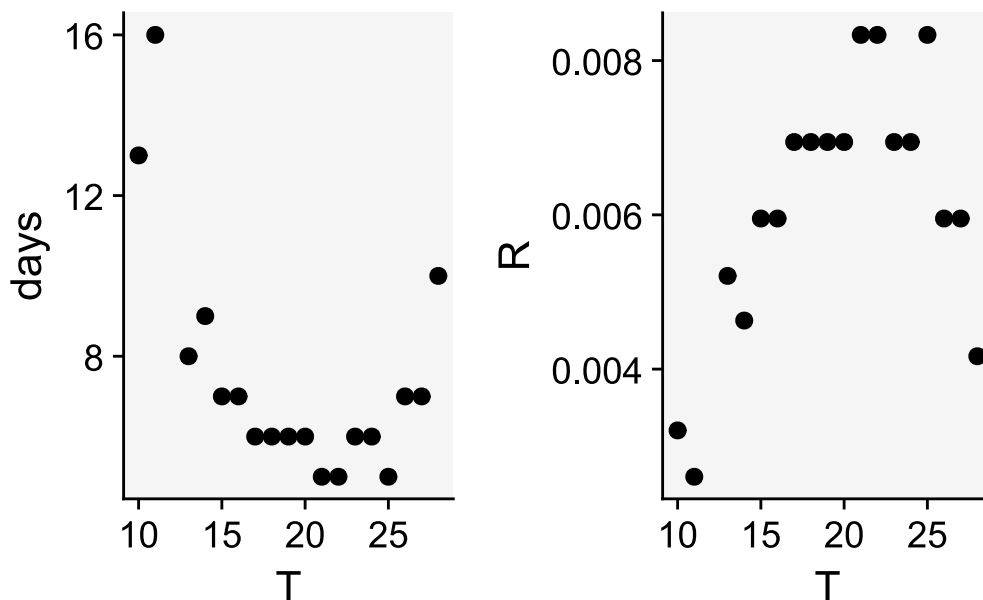


Figure 21.12: Latent period (left) and rate of development, the inverse of the latent period (right), of powdery mildew at constant temperatures

We will fit the parabolic function proposed by [Bernard et al. \(2013\)](#) which predicts a thermal response curve (developmental rate, R), which is the relationship between the inverse of latent period and temperature. We need to enter the values for optimum temperature (where latent period is shortest) and the minimum latent period. The model is given by:

$$R(T) = \frac{k}{LP_{\min} + a \times (T - T_{\text{opt}})^2}$$

```

# Load necessary package
#library(minpack.lm)

# Define the model formula
# model_formula <- R ~ (a + b * T)^(c * T)
LPmin <- 5 # minimum latent period

```

```

Topt <- 21 # Optimal temperature
model_formula2 <- R ~ k / (LPmin + a * (T - Topt)^2)

# Set initial parameter estimates
#start_values <- list(a = 0.1, b = 0.01, c = 0.01)
start_values2 <- list(a = 0.1, k = 1)
# Fit the model
#fit_rate <- nlsLM(model_formula, data = latent_rate, start = start_values)
fit_rate2 <- nls(model_formula2, data = latent_rate, start = start_values2)

# View the summary of the fit
summary(fit_rate2)

```

```

Formula: R ~ k/(LPmin + a * (T - Topt)^2)

Parameters:
      Estimate Std. Error t value Pr(>|t|)
a 0.060274    0.010705    5.63 3.76e-05 ***
k 0.039520    0.001464   26.99 9.05e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0006936 on 16 degrees of freedom

Number of iterations to convergence: 6
Achieved convergence tolerance: 3.165e-06

```

```
fit_rate2$m$getAllPars()
```

```

      a      k
0.06027417 0.03952035

```

```

a <- fit_rate2$m$getAllPars()[1]
k <- fit_rate2$m$getAllPars()[2]

```

Now we reproduce the plot with the fitted data. Note that the curve is not the same shown in the paper because we used a different equation.

```

T <- seq(10, 29, 0.1)
#R <- (a + b * T)^(c * T)
R <- k / (LPmin + a * (T - Topt)^2)
dat2 <- data.frame(T, R)

dat2 |>
  ggplot(aes(T, R)) +
  geom_line() +
  geom_point(data = latent_rate, aes(T, R)) +
  theme_r4pde(font_size = 16) +
  labs(x = "Temperature", y = "Inverse of the latent period (hour -1)",
       title = "")

```

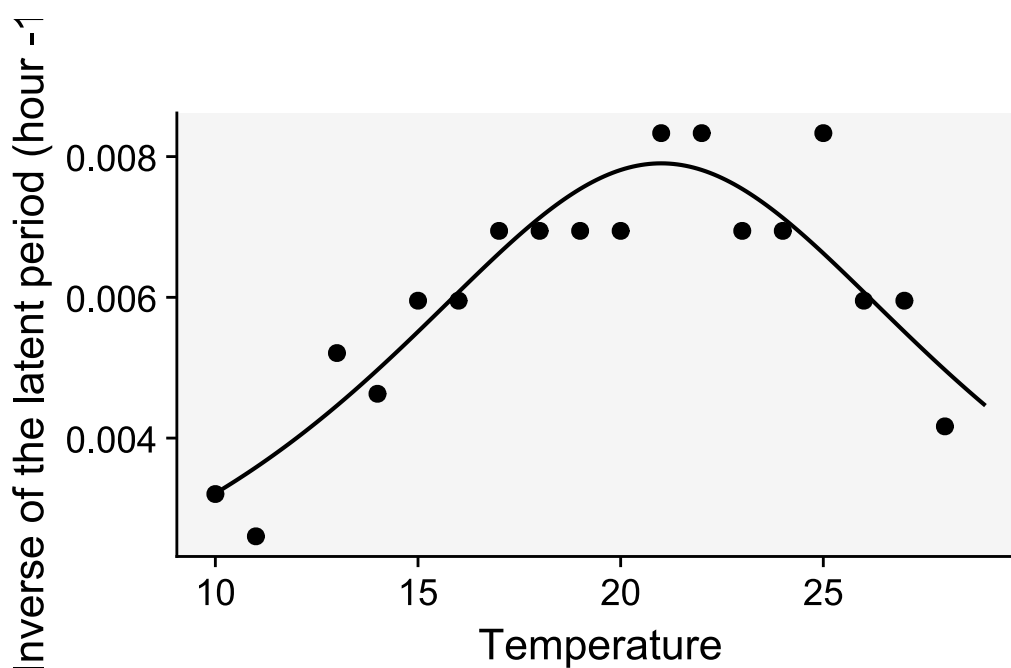



Figure 21.13: Rate of development (inverse of the latent period) as a function of temperature; the line is the fitted parabolic function

Field data

While pathogen inoculum, host resistance, and agronomic factors are sometimes included alongside weather in empirically derived models using field data (Cao et al., 2015; Mehra et al., 2017; Shah et al., 2013), only a few models explicitly incorporate non-weather factors (Mehra et al., 2016; Paul & Munkvold, 2004). In most cases, these models primarily rely on weather variables as predictors (González-Domínguez et al., 2023). This focus reflects the critical role of weather in driving key processes in the disease cycle, such as pathogen survival, dispersal, host infection, and reproduction (De Wolf & Isard, 2007). Consequently, a primary objective for plant epidemiologists is to identify and quantify the relationships between weather conditions and measures of disease intensity (Coakley, 1988; Coakley et al., 1988; Del Ponte et al., 2006; El Jarroudi et al., 2017; Pietravalle et al., 2003; Shah et al., 2013; 2019a).

In modeling efforts, the disease variable can be represented either as a continuous measure (e.g., incidence or severity) or as categorical data, which may be binary (e.g., non-epidemic vs. epidemic) or multinomial (e.g., low, moderate, and high severity). This variability in response types informs the selection of suitable modeling techniques, ensuring that the model accurately captures the nature of the data and the relationships between weather variables and disease outcomes.

In this section, we demonstrate several modeling approaches that can be applied when field data is available. These examples will cover a range of techniques, starting with variable construction, which involves transforming raw weather data into summary measures that can effectively represent conditions relevant to disease outcomes. Next, variable selection methods will be explored to identify the most influential predictors, ensuring that models are both accurate and interpretable. The focus will then shift to model fitting, showing how different models, such as linear and logistic regression, can be used to capture relationships between weather variables and disease endpoints. Finally, model evaluation will be addressed,

emphasizing metrics like accuracy, sensitivity, and area under the curve (AUC), which are crucial for assessing the predictive performance and reliability of the models developed.

Below is a typical workflow for developing a disease prediction model, starting with disease data collection and weather data retrieval. The process includes data pre-processing, feature engineering, variable selection, model fitting, cross-validation, model tuning, and evaluation, followed by interpretation. Iterative feedback between model evaluation and variable selection aims to optimize model performance.

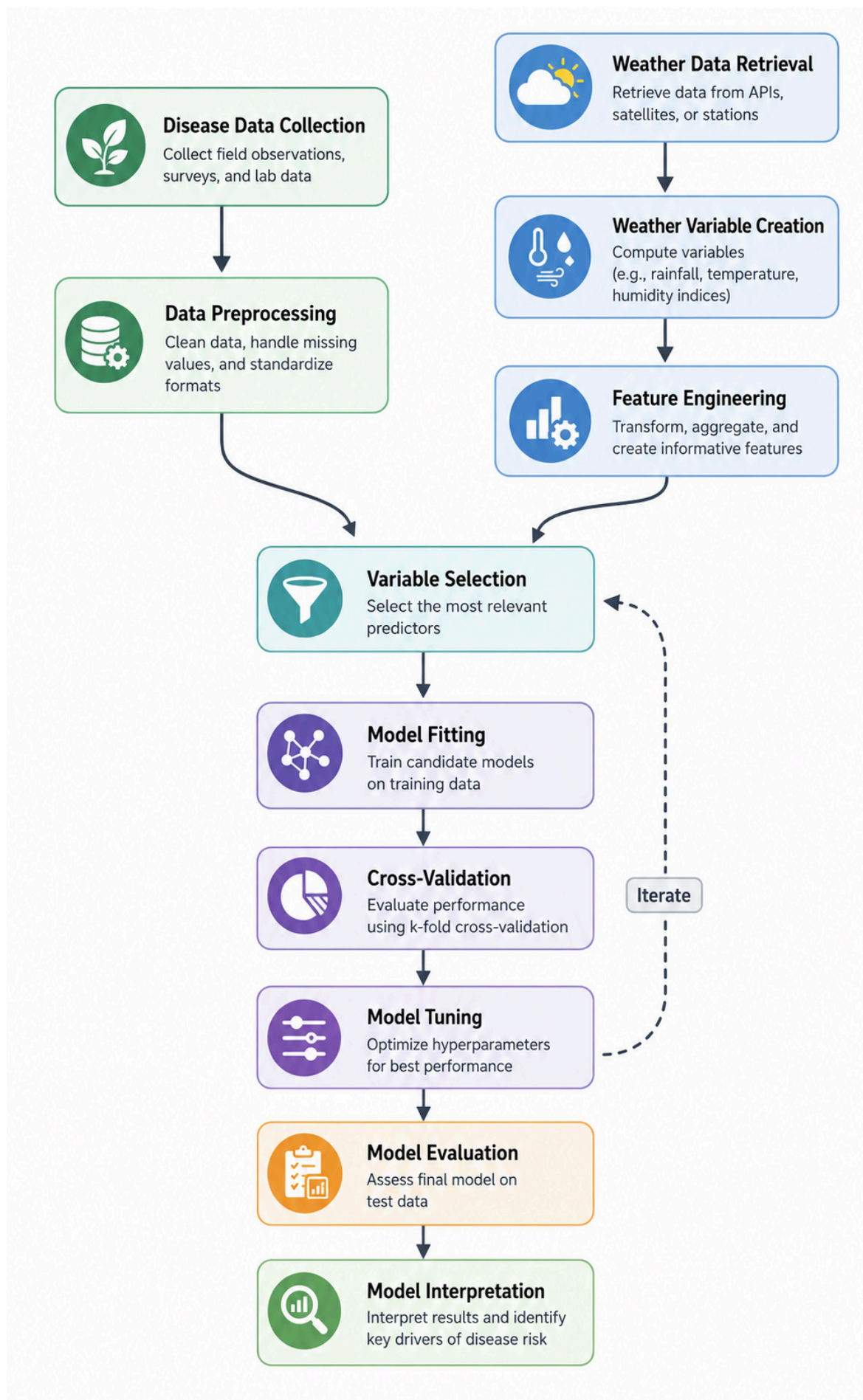


Figure 21.14: Key steps for developing a plant disease prediction model

Variable creation and selection

Variable construction, particularly for weather-related variables, involves not only data transformation methods but also requires an understanding of how diseases respond to specific weather conditions at particular growth stages (De Cól et al., 2024; De Wolf et al., 2003). This approach ensures that the variables derived accurately capture biologically relevant processes, improving the realism and relevance of the model inputs.

In addition, data mining techniques are employed to systematically explore time-series data and identify potential weather-disease relationships (Coakley et al., 1988; Pietravalle et al., 2003; Shah et al., 2019a). These techniques involve creating lagged variables, moving averages, or window-based summaries that capture delayed or cumulative effects of weather on disease outcomes. By integrating system knowledge with data mining, researchers aim to construct variables that are both biologically meaningful and statistically robust, improving the chances of identifying predictors that enhance model accuracy and interpretability.

Window-pane

Variable construction

With regards to weather variable creation and selection for data-mining purposes, window-pane analysis, first introduced in the mid-1980s (Coakley, 1985), has been widely used in modeling studies in plant pathology (Calvero Jr et al., 1996; Coakley, 1988; Coakley et al., 1988; Dalla Lana et al., 2021a; Gouache et al., 2015; Kriss et al., 2010; Pietravalle et al., 2003; Te Beest et al., 2008). This method aids in identifying weather conditions that are most strongly associated with disease outcomes by segmenting a continuous time series (e.g. daily temperature, relative humidity, and rainfall), into discrete, fixed-length windows.

The analysis involves summarizing conditions within each window (e.g., mean, sum, count) and correlating these summaries with disease outcomes, which may be expressed as continuous measures (e.g., severity) or as categorical variables (e.g., low vs. high levels). This approach allows users to set specific start and end times, as well as window lengths, enabling the exploration of different temporal relationships between weather and disease. By sliding the start and end points along the series, multiple overlapping windows are generated, making it possible to identify the most informative variables for modeling. The selected optimal fixed-time and fixed-window-length variables derived from this analysis serve as predictors in model development, helping to improve the accuracy and relevance of disease forecasting models.

Here's an R code that demonstrates how the windows are defined over a 28-day period using four fixed window lengths (7, 14, 21, and 28 days), generating a total of 46 variables.

```
library(dplyr)
library(ggplot2)

# Define total days and window lengths
max_days <- 28
window_lengths <- c(7, 14, 21, 28)

# Create an empty data frame for all sliding windows
window_data <- data.frame()

# Populate the data frame with start and end points for each window
var_id <- 1 # Variable ID for each window
```

```

for (length in sort(window_lengths)) { # Sort window lengths from shortest to longest
  for (start_day in 0:(max_days - length)) {
    end_day <- start_day + length
    window_data <- rbind(
      window_data,
      data.frame(
        start = start_day,
        end = end_day,
        var_id = var_id,
        window_length = length
      )
    )
    var_id <- var_id + 1 # Increment variable ID
  }
}

# Convert window_length to a factor for correct ordering in the legend
window_data$window_length <- factor(window_data$window_length, levels =
  sort(unique(window_data$window_length)))

```

```

window_data |>
  ggplot(aes(x = start, xend = end, y = var_id, yend = var_id)) +
  geom_segment(linewidth = 2) + # Line segments for each window
  scale_x_continuous(breaks = 0:max_days, limits = c(0, max_days)) +
  scale_y_continuous(breaks = 1:var_id) +
  labs(title = "Window-pane",
       subtitle = "Each variable of 7, 14, 21 and 28 days length over 28 days",
       x = "Days", y = "Variable ID", color = "Window Length (days)") +
  r4pde::theme_r4pde(font_size = 14) +
  theme(legend.position = "right")

```

Ignoring unknown labels:

- colour : "Window Length (days)"

Window-pane

Each variable of 7, 14, 21 and 28 days length over 28 days

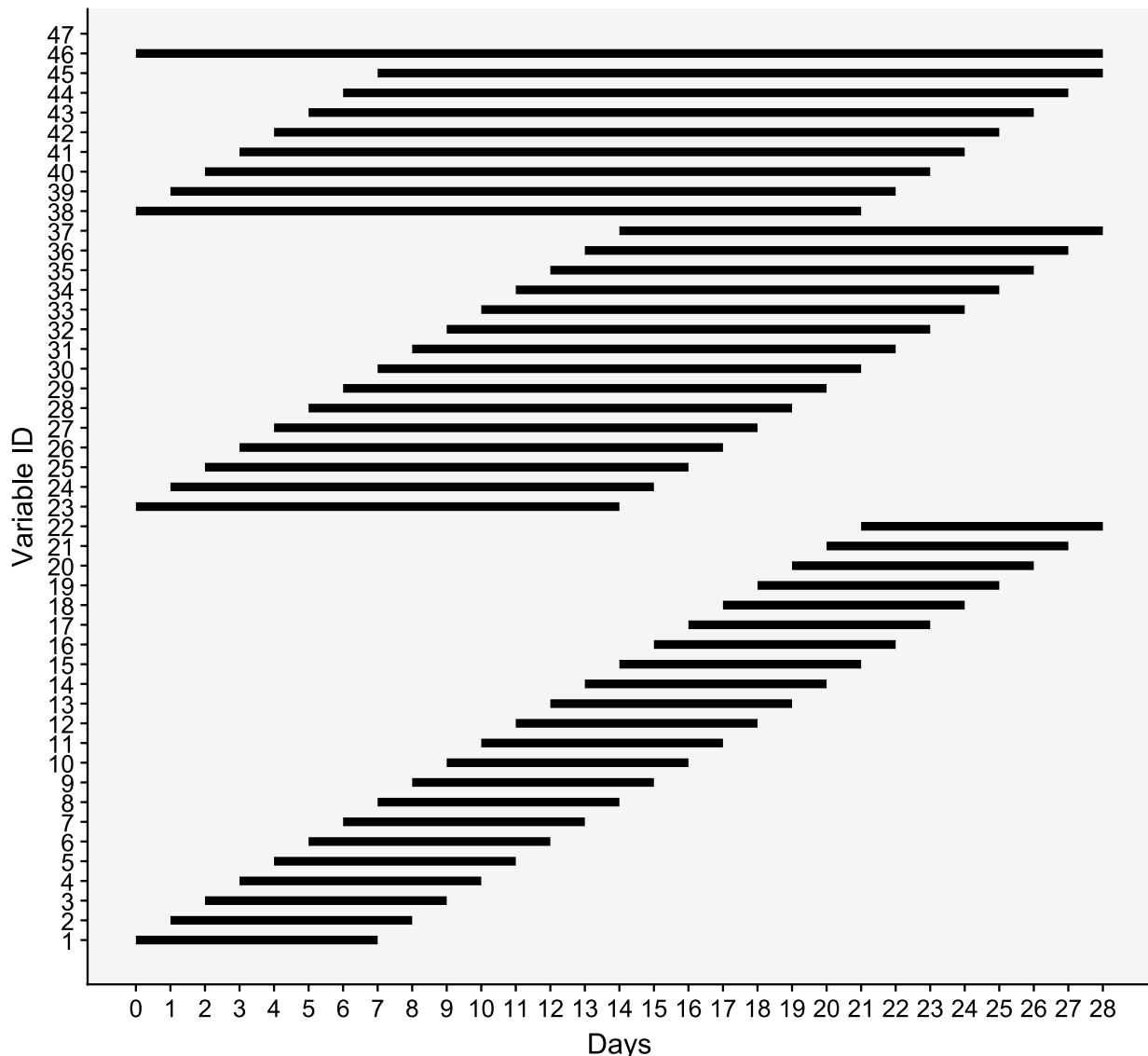


Figure 21.15: Fixed-length windows (7, 14, 21 and 28 days) generated over a 28-day period; each segment is one window-pane variable

The window-pane analysis requires a spreadsheet program (Kriss et al., 2010) or a specific algorithm that automates the creation of sliding windows at defined starting and ending times relative to a reference date. In the seminal work, software was programmed in the FORTRAN language and named WINDOW (Coakley, 1985). It enabled the creation of windows and the calculation of summary statistics, including correlation with disease severity. Building on the original idea, a Genstat 6.1 algorithm was developed in the early 2000s, incorporating further adjustments such as the implementation of bootstrapping analysis to validate correlations and misclassifications identified by window-pane (Pietravalle et al., 2003; Te Beest et al., 2008). More recently, window-pane analysis including variable creation and analysis has been conducted in R using custom-made scripts (Dalla Lana et al., 2021a; Gouache et al., 2015).

I will demonstrate the `windowpane()` function of the `{r4pde}` package developed to facilitate the creation of variables using the window-pane approach. First, let's load a dataset that contains

information on the disease, as well as metadata, including a key date that will be used as the starting point for window creation. The BlastWheat dataset, which is included in the {r4pde} package, was provided by De Cól et al. (2024).

```
library(r4pde)
library(dplyr)
trials <- BlastWheat
glimpse(trials)
```

```
Rows: 143
Columns: 10
$ study      <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ...
$ year       <dbl> 2012, 2012, 2012, 2012, 2012, 2013, 2013, 2013, 2013, 2014,...
$ location   <chr> "Dourados", "Palotina", "Londrina", "Planaltina", "Itaberá"...
$ state      <chr> "MS", "PR", "PR", "DF", "SP", "MS", "MG", "DF", "SP", "MG",...
$ latitude   <dbl> -22.27516, -24.35483, -23.35916, -15.60387, -24.06832, -22....
$ longitude  <dbl> -54.81640, -53.75794, -51.16476, -47.71381, -49.15575, -54....
$ heading    <chr> "10-05-2012", "09-06-2012", "01-06-2012", "03-05-2012", "15...
$ inc_mean   <dbl> 93.30, 40.50, 100.00, 35.50, 7.71, 48.70, 22.00, 46.50, 68....
$ index_mean <dbl> 68.65, 11.73, 92.86, 6.36, 0.35, 9.17, 2.91, 10.41, 41.86, ...
$ yld_mean   <dbl> 1097.00, 1219.38, 495.12, 1747.44, 2148.31, 506.00, 1562.75...
```

We can note that the heading date, which will be used as reference date in our analysis, is not defined as date object, which needs correction.

```
trials$heading = as.Date(trials$heading, format = "%d-%m-%Y")
glimpse(trials$heading)
```

```
Date[1:143], format: "2012-05-10" "2012-06-09" "2012-06-01" "2012-05-03" "2012-04-15" ...
```

The weather data for our analysis will be downloaded from NASA POWER. Since we have multiple trials with different heading dates, a wrapper function for the `get_power()` function from the {nasapower} package was created, included in {r4pde}, and named `get_nasapower()`. This function enables the download of data for a period “around” the key date, which can be defined by the user. In our case, we will download data from 28 days before and 28 days after the heading date.

```
weather_data <- get_nasapower(
  data = trials,
  days_around = 28,
  date_col = "heading",
  pars = c("T2M", "T2M_MAX", "T2M_MIN", "T2M_RANGE", "RH2M",
           "PRECTOTCORR", "T2MDEW", "WS2M", "PS", "GWETTOP",
           "GWETPROF", "CLRSKY_SFC_PAR_TOT")
)
# See all parameters in the website: https://power.larc.nasa.gov/#resources

# save the data for faster rendering
write_csv(weather_data, "data/weather_windowpane.csv")
```

Now we can see the weather data and join the two dataframes.

```
# read the data
weather_data <- readr::read_csv("data/weather_windowpane.csv")
```



```
glimpse(weather_data)
```

```
Rows: 8,151
Columns: 14
$ LON      <dbl> -54.8164, -54.8164, -54.8164, -54.8164, -54.8164, -54.8164...
$ LAT      <dbl> -22.27516, -22.27516, -22.27516, -22.27516, -22.27516, -22...
$ YEAR     <dbl> 2012, 2012, 2012, 2012, 2012, 2012, 2012, 2012, 2012, 2012...
$ MM       <dbl> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5...
$ DD       <dbl> 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26...
$ DOY      <dbl> 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114...
$ YYYYMMDD <date> 2012-04-12, 2012-04-13, 2012-04-14, 2012-04-15, 2012-04-1...
$ T2M      <dbl> 24.31, 25.83, 24.99, 24.15, 22.72, 23.33, 23.76, 24.83, 24...
$ RH2M     <dbl> 83.69, 77.62, 81.69, 82.69, 74.62, 74.44, 70.19, 67.75, 77...
$ PRECTOTCORR <dbl> 0.02, 1.13, 11.00, 0.06, 0.00, 0.04, 0.00, 0.03, 24.08, 20...
$ T2M_MAX  <dbl> 28.61, 31.01, 30.01, 29.43, 30.33, 30.69, 31.51, 32.49, 30...
$ T2M_MIN  <dbl> 20.82, 20.43, 21.11, 19.04, 16.31, 17.37, 17.15, 17.69, 20...
$ T2MDEW   <dbl> 21.19, 21.23, 21.38, 20.75, 17.47, 18.01, 17.29, 17.85, 20...
$ study    <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
```

```
# apply a full join
trials_weather <- full_join(trials, weather_data)
```

We are now ready to use the `windowpane` function to create new variables. The function has several arguments. Note the two date variables: `end_date`, which serves as the reference for sliding the windows, and `date_col`, which represents the date for each day in the time series. The `summary_type` specifies the statistic to be calculated, while the `direction` determines whether the sliding windows will move backward, forward, or in both directions relative to the end date. Lastly, the `group_by` argument specifies the index variable for the epidemic or study.

We will create new variables based on the mean daily temperature (T2M), with each variable representing the mean value over one of the four window lengths (7, 14, 21, and 28 days) defined in the `window_length` argument. We will only generate variables that cover periods before the heading date, using “backward” in the `direction` argument.

```
# Create window variables separated for each weather variable
```

```
wp_T2M <- windowpane(
  data = trials_weather,
  end_date_col = heading,
  date_col = YYYYMMDD,
  variable = T2M,
  summary_type = "mean",
  threshold = NULL,
  window_lengths = c(7, 14, 21, 28),
  direction = "backward",
  group_by_cols = "study",
)
```

```
wpT2M_MIN_15 <- windowpane(
  data = trials_weather,
  end_date_col = heading,
  date_col = YYYYMMDD,
  variable = T2M_MIN,
  summary_type = "below_threshold",
```



```

    threshold = 15,
    window_lengths = c(7, 14, 21, 28),
    direction = "backward",
    group_by_cols = "study",
  )

wpT2M_MIN <- windowpane(
  data = trials_weather,
  end_date_col = heading,
  date_col = YYYYMMDD,
  variable = T2M_MIN,
  summary_type = "mean",
  threshold = NULL,
  window_lengths = c(7, 14, 21, 28),
  direction = "backward",
  group_by_cols = "study",
)

wpT2M_MAX <- windowpane(
  data = trials_weather,
  end_date_col = heading,
  date_col = YYYYMMDD,
  variable = T2M_MAX,
  summary_type = "mean",
  threshold = NULL,
  window_lengths = c(7, 14, 21, 28),
  direction = "backward",
  group_by_cols = "study",
)

wpPREC <- windowpane(
  data = trials_weather,
  end_date_col = heading,
  date_col = YYYYMMDD,
  variable = PRECTOTCORR,
  summary_type = "sum",
  threshold = NULL,
  window_lengths = c(7, 14, 21, 28),
  direction = "backward",
  group_by_cols = "study",
)

wpRH2M <- windowpane(
  data = trials_weather,
  end_date_col = heading,
  date_col = YYYYMMDD,
  variable = RH2M,
  summary_type = "mean",
  threshold = NULL,
  window_lengths = c(7, 14, 21, 28),
  direction = "backward",
  group_by_cols = "study",
)

# combine all datasets
wp_all <- cbind(wp_T2M, wpT2M_MIN_15, wpT2M_MIN, wpT2M_MAX, wpPREC, wpRH2M)

```

Correlations and multiple hypothesis test

The window-pane analysis begins by quantifying the associations between each summary weather variable and disease response using a specific correlation coefficient (Pearson or Spearman). Usually, Spearman's rank correlation can be preferred due to its ability to measure monotonic relationships and its robustness to outliers. In other cases, Spearman was used because the disease data was ordinal (Kriss et al., 2010).

In a recent study, Dalla Lana et al. (2021a), differing from Kriss et al. (2010), proposed the estimation of the precision of these correlations via bootstrapping, where a high number of samples (e.g. 1000) are randomly drawn (with replacement) from the original dataset. For each bootstrap sample, correlations between weather variables and disease outcomes are calculated, and the average across samples is used as the final measure of association. This approach ensures a more reliable estimation of the correlations by capturing variability and improving statistical robustness.

The window-pane analysis involves numerous tests, as each time window generates a separate test statistic. Because many tests are conducted, the global significance level becomes higher than the critical significance level set for individual tests, increasing the risk of false positives (Kriss et al., 2010). Additionally, the correlations among test statistics are influenced by overlapping time windows, shared data, and large-scale climatic patterns. To address this issue, Kriss et al. (2010) proposed the use of the Simes' method, which tests the global null hypothesis that none of the individual correlations are significant. Simes' method orders p-values and rejects the global null hypothesis if any adjusted p-value meets a specific threshold.

While this method indicates whether at least one correlation is significant, it does not provide significance for individual correlations. Therefore, the authors proposed that the individual correlation coefficients should be compared against a more stringent significance level ($\alpha = 0.005$ instead of 0.05), reducing the likelihood of false positives but increasing false negatives. Although this adjustment is independent of the Simes' method, there was a general alignment: significant global results often corresponded to significant individual correlations, and non-significant global results typically lacked significant individual correlations (Kriss et al., 2010).

Let's calculate the bootstrapped correlation coefficients combined with the Simes' method. First, we need to subset our variables to the specific combination of weather and window.

```
T2M_MIN_7 = wp_all |> dplyr::select(contains("length7_T2M_MIN_mean"))
T2M_MIN_14 = wp_all |> dplyr::select(contains("length14_T2M_MIN_mean"))
T2M_MIN_21 = wp_all |> dplyr::select(contains("length21_T2M_MIN_mean"))
T2M_MIN_28 = wp_all |> dplyr::select(contains("length28_T2M_MIN_mean"))

T2M_MAX_7 = wp_all |> dplyr::select(contains("length7_T2M_MAX_mean"))
T2M_MAX_14 = wp_all |> dplyr::select(contains("length14_T2M_MAX_mean"))
T2M_MAX_21 = wp_all |> dplyr::select(contains("length21_T2M_MAX_mean"))
T2M_MAX_28 = wp_all |> dplyr::select(contains("length28_T2M_MAX_mean"))

T2M_7 = wp_all |> dplyr::select(contains("length7_T2M_mean"))
T2M_14 = wp_all |> dplyr::select(contains("length14_T2M_mean"))
T2M_21 = wp_all |> dplyr::select(contains("length21_T2M_mean"))
T2M_28 = wp_all |> dplyr::select(contains("length28_T2M_mean"))

RH2M_7 = wp_all |> dplyr::select(contains("length7_RH2M_mean"))
RH2M_14 = wp_all |> dplyr::select(contains("length14_RH2M_mean"))
RH2M_21 = wp_all |> dplyr::select(contains("length21_RH2M_mean"))
```

```

RH2_28 = wp_all |> dplyr::select(contains("length28_RH2M_mean"))

PRECTOTCORR_7 = wp_all |> dplyr::select(contains("length7_PRECTOTCORR"))
PRECTOTCORR_14 = wp_all |> dplyr::select(contains("length14_PRECTOTCORR"))
PRECTOTCORR_21 = wp_all |> dplyr::select(contains("length21_PRECTOTCORR"))
PRECTOTCORR_28 = wp_all |> dplyr::select(contains("length28_PRECTOTCORR"))

T2M_MINb_7 = wp_all |> dplyr::select(contains("length7_T2M_MIN_below"))
T2M_MINb_14 = wp_all |> dplyr::select(contains("length14_T2M_MIN_below"))
T2M_MINb_21 = wp_all |> dplyr::select(contains("length21_T2M_MIN_below"))
T2M_MINb_28 = wp_all |> dplyr::select(contains("length28_T2M_MIN_below"))

```

Now, we can use the `windowpane_tests()` function from the `{r4pde}` package to analyze each of the datasets created above. This function will compute the bootstrapped correlation coefficients for the variables of interest and apply the Simes' procedure to account for multiple testing, providing adjusted P-values for more robust statistical inference.

```

library(boot)
data <- T2M_MINb_7
data$inc <- trials$inc_mean
response_var <- 'inc'
results <- windowpane_tests(data, response_var, corr_type = "spearman", R = 1000)
results_TMINb <- results$results

results

```

```

$results

```

	variable	correlation	p_value				
1	bw_length7_T2M_MIN_below_threshold_-18_-24	-0.5295915	1.040432e-11				
2	bw_length7_T2M_MIN_below_threshold_-17_-23	-0.5136581	5.359609e-11				
3	bw_length7_T2M_MIN_below_threshold_-15_-21	-0.5116609	6.543248e-11				
4	bw_length7_T2M_MIN_below_threshold_-10_-16	-0.5063934	1.100732e-10				
5	bw_length7_T2M_MIN_below_threshold_-11_-17	-0.5054218	1.210402e-10				
6	bw_length7_T2M_MIN_below_threshold_-16_-22	-0.5038720	1.407514e-10				
7	bw_length7_T2M_MIN_below_threshold_-12_-18	-0.4972482	2.659657e-10				
8	bw_length7_T2M_MIN_below_threshold_-19_-25	-0.4951918	3.231718e-10				
9	bw_length7_T2M_MIN_below_threshold_-9_-15	-0.4913413	4.638201e-10				
10	bw_length7_T2M_MIN_below_threshold_-7_-13	-0.4882483	6.180090e-10				
11	bw_length7_T2M_MIN_below_threshold_-3_-9	-0.4793727	1.386299e-09				
12	bw_length7_T2M_MIN_below_threshold_0_-6	-0.4757057	1.922737e-09				
13	bw_length7_T2M_MIN_below_threshold_-8_-14	-0.4743890	2.160334e-09				
14	bw_length7_T2M_MIN_below_threshold_-14_-20	-0.4742346	2.189972e-09				
15	bw_length7_T2M_MIN_below_threshold_-4_-10	-0.4735015	2.336183e-09				
16	bw_length7_T2M_MIN_below_threshold_-2_-8	-0.4730773	2.425031e-09				
17	bw_length7_T2M_MIN_below_threshold_-1_-7	-0.4698400	3.218863e-09				
18	bw_length7_T2M_MIN_below_threshold_-6_-12	-0.4663226	4.364062e-09				
19	bw_length7_T2M_MIN_below_threshold_-13_-19	-0.4630281	5.785628e-09				
20	bw_length7_T2M_MIN_below_threshold_-5_-11	-0.4594875	7.807700e-09				
21	bw_length7_T2M_MIN_below_threshold_-20_-26	-0.4334293	6.401965e-08				
22	bw_length7_T2M_MIN_below_threshold_-21_-27	-0.4291061	8.925631e-08				
	mean_corr	sd_corr	median_corr	rank	m	simes_threshold	significant_simes
1	-0.5268631	0.05884301	-0.5274600	1	22	0.002272727	TRUE
2	-0.5096147	0.06208830	-0.5118708	2	22	0.004545455	TRUE
3	-0.5108678	0.05905196	-0.5132224	3	22	0.006818182	TRUE
4	-0.5021705	0.06610590	-0.5058771	4	22	0.009090909	TRUE
5	-0.5014453	0.06705970	-0.5041323	5	22	0.011363636	TRUE

```

6 -0.4997765 0.06399862 -0.5048758 6 22 0.013636364 TRUE
7 -0.4973725 0.06329604 -0.4993432 7 22 0.015909091 TRUE
8 -0.4930485 0.05988090 -0.4933091 8 22 0.018181818 TRUE
9 -0.4941782 0.06485541 -0.4963972 9 22 0.020454545 TRUE
10 -0.4900910 0.06679479 -0.4948681 10 22 0.022727273 TRUE
11 -0.4790248 0.06322360 -0.4813064 11 22 0.025000000 TRUE
12 -0.4719840 0.06608034 -0.4743389 12 22 0.027272727 TRUE
13 -0.4728241 0.06632855 -0.4725078 13 22 0.029545455 TRUE
14 -0.4712244 0.06707354 -0.4725337 14 22 0.031818182 TRUE
15 -0.4688074 0.06803790 -0.4723670 15 22 0.034090909 TRUE
16 -0.4715528 0.06465161 -0.4748049 16 22 0.036363636 TRUE
17 -0.4699089 0.06544775 -0.4725348 17 22 0.038636364 TRUE
18 -0.4635897 0.06749465 -0.4656357 18 22 0.040909091 TRUE
19 -0.4649800 0.06801518 -0.4681421 19 22 0.043181818 TRUE
20 -0.4527452 0.06742145 -0.4535339 20 22 0.045454545 TRUE
21 -0.4316172 0.06527766 -0.4348546 21 22 0.047727273 TRUE
22 -0.4298104 0.06904030 -0.4346586 22 22 0.050000000 TRUE

individual_significant
1 TRUE
2 TRUE
3 TRUE
4 TRUE
5 TRUE
6 TRUE
7 TRUE
8 TRUE
9 TRUE
10 TRUE
11 TRUE
12 TRUE
13 TRUE
14 TRUE
15 TRUE
16 TRUE
17 TRUE
18 TRUE
19 TRUE
20 TRUE
21 TRUE
22 TRUE

$summary_table
      Metric      Value
1 Global P-value (Pg) 2.288950e-10
2 Max Correlation 5.295915e-01

$global_significant
[1] TRUE

```

The window-pane tests show negative correlations between disease incidence and the number of days with TMIN below 15 °C in the 7-day windows. The mean correlations range from -0.53 to -0.43.

The global P-value in the output accounts for multiple testing with Simes' procedure. A low global P-value indicates that at least one window is significantly associated with the response. Together, these results suggest that cold conditions in the weeks before heading are associated with a lower incidence of wheat blast.

Let's apply the test function to 7-day long windows for three other variables: relative humidity, precipitation and maximum temperature.

```
library(r4pde)
data1 <- RH2M_7
data1$inc <- trials$inc_mean
response_var <- 'inc'
results1 <- windowpane_tests(data1, response_var, corr_type = "spearman", R = 1000)
results_RH <- results1$results

data2 <- PRECTOTCORR_7
data2$inc <- trials$inc_mean
response_var <- 'inc'
results2 <- windowpane_tests(data2, response_var, corr_type = "spearman", R = 1000)
results_PREC <- results2$results

data3 <- T2M_MAX_7 # enter the dataset
data3$inc <- trials$inc_mean
response_var <- 'inc'
results3 <- windowpane_tests(data3, response_var, corr_type = "spearman", R = 1000)
results_TMAX <- results3$results
```

The first variable in the results dataframe contains the name of the variable. We can extract the first and last day of each window (relative to the heading date) from the variable name using the `extract` function, which allows for regex-based extraction into new columns. We then keep the earliest day of each window as its starting day (`start`). The column names created by `windowpane()` start with the direction of the windows (`bw_` for backward), so we select variables with `contains()` instead of `starts_with()`.

```
# Use strcapture to extract components into new columns
df_TMINb7 <- results_TMINb |>
  extract(variable, into = c("variable_prefix", "low", "high"),
          regex = "^(.*)_(-?\\d+)_(-?\\d+)$", convert = TRUE) |>
  dplyr::mutate(start = pmin(low, high))

df_RH7 <- results_RH |>
  extract(variable, into = c("variable_prefix", "low", "high"),
          regex = "^(.*)_(-?\\d+)_(-?\\d+)$", convert = TRUE) |>
  dplyr::mutate(start = pmin(low, high))

df_PREC7 <- results_PREC |>
  extract(variable, into = c("variable_prefix", "low", "high"),
          regex = "^(.*)_(-?\\d+)_(-?\\d+)$", convert = TRUE) |>
  dplyr::mutate(start = pmin(low, high))

df_TMAX7 <- results_TMAX |>
  extract(variable, into = c("variable_prefix", "low", "high"),
          regex = "^(.*)_(-?\\d+)_(-?\\d+)$", convert = TRUE) |>
  dplyr::mutate(start = pmin(low, high))
```

Now we can plot the mean correlation for the start day of each window.

```

p_TMIN <- df_TMINb7 |>
  ggplot(aes(start, mean_corr, fill = p_value))+
  ylim(-1, 1)+
  geom_col()+
  theme_r4pde(font_size = 12)+
  labs(title = "N. days TMIN < 15C",
        subtitle = "7-day window",
        y = "Spearman's correlation",
        x = "Day relative to heading date ")

p_RH <- df_RH7 |>
  ggplot(aes(start, mean_corr, fill = p_value))+
  ylim(-1, 1)+
  geom_col()+
  theme_r4pde(font_size = 12)+
  labs(title = "Relative humidity",
        subtitle = "7-day window",
        y = "Spearman's correlation",
        x = "Day relative to heading date ")

p_TMAX <- df_TMAX7 |>
  ggplot(aes(start, mean_corr, fill = p_value))+
  ylim(-1, 1)+
  geom_col()+
  theme_r4pde(font_size = 12)+
  labs(title = "Max. temperature",
        subtitle = "7-day window",
        y = "Spearman's correlation",
        x = "Day relative to heading date ")

p_PREC <- df_PREC7 |>
  ggplot(aes(start, mean_corr, fill = p_value))+
  ylim(-1, 1)+
  geom_col()+
  theme_r4pde(font_size = 12)+
  labs(title = "Cumulative rainfall",
        subtitle = "7-day window",
        y = "Spearman's correlation",
        x = "Day relative to heading date ")

scale_common <- scale_fill_viridis_c(limits = c(0, 1),
                                     na.value = "grey90")

p_TMIN <- p_TMIN + scale_common
p_TMAX <- p_TMAX + scale_common
p_RH <- p_RH + scale_common
p_PREC <- p_PREC + scale_common

# Combine plots using patchwork and collect guides
(p_TMIN | p_TMAX) / (p_RH | p_PREC) +
  plot_layout(guides = "collect")

```

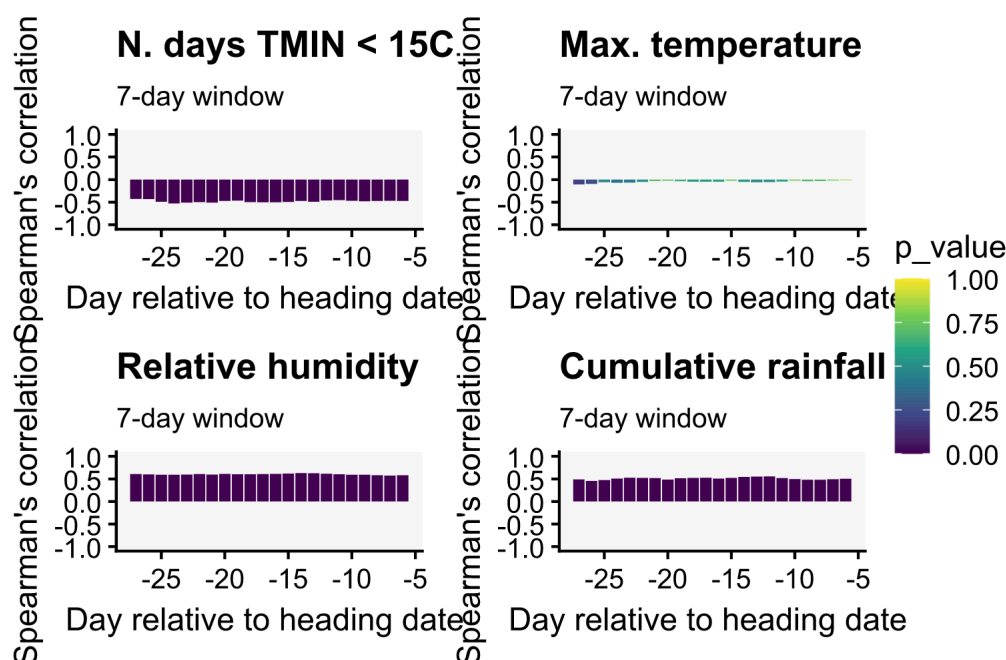


Figure 21.16: Mean Spearman correlation between wheat blast incidence and 7-day window-pane variables, by starting day of the window relative to heading; the fill indicates the P-value

Interpretation. All 7-day-length variables but maximum temperature were strongly associated with the incidence of wheat blast. These three variables can be used as candidates for developing prediction models.

Functional data analysis

Functional Data Analysis (FDA) is a statistical approach used to analyze and model data that varies continuously over time or space. It represents entire data sequences (e.g., time series) as smooth functions, allowing for the exploration of trends, patterns, and relationships in complex, continuous processes rather than relying on discrete, segmented data points (Gertheiss et al., 2024; Ramsay & Silverman, 2005).

In the FDA framework, scalar-on-function and function-on-scalar models are two types of FDA models, designed to examine relationships between functional predictors and outcomes. Function-on-scalar models use single values (e.g., mean daily relative humidity) as predictors to explain the shape of an entire outcome curve (e.g. disease). Meanwhile, in scalar-on-function models, time series data (e.g., daily temperatures) are used as predictors to identify which parts of the series are linked to a single outcome. Together, these FDA models offer complementary insights, allowing either for predicting a scalar outcome from curves or understanding how scalar predictors influence entire functions (Ramsay & Silverman, 2005).

Initially applied in plant pathology to study the relationship between *Fusarium* head blight outbreaks and weather variables over time (using the function-on-scalar model) (Shah et al., 2019b), FDA has since been adopted in other plant pathology studies with similar objectives (Alves et al., 2022; Hjelkrem et al., 2021; Shah et al., 2019a).

In particular, Shah et al. (2019a) used FDA as a basis to develop logistic regression models for large-scale deployment by utilizing FDA through a scalar-on-function approach to predict plant diseases - in this approach, the model identifies which parts of the time series (e.g., certain weeks or days) are most associated with the disease outcome, helping to predict

whether an epidemic is likely to occur based on patterns in the weather series. Moreover, FDA was employed to identify the most relevant temporal regions of weather series associated with disease outbreaks, improving prediction by summarizing key windows across critical growth stages. The authors discussed that, unlike traditional window-pane analysis, FDA reduces statistical testing issues and offers a more refined method for incorporating novel predictors into simple, effective models (Shah et al., 2019a).

With the objective of identifying temporal regions of the weather series, Alves et al. (unpublished) proposed the use of a function-on-scalar model to gain insights about the time series related to a binary disease outcome by identifying significant regions in the weather series where the curves diverge. The function-on-scalar approach here compares the average functional trajectories of two groups: epidemic vs. non-epidemic. It identifies time regions where the mean weather curves for the two groups differ significantly. While the model doesn't directly predict a binary response, it aids in diagnosing critical time points in the series that differentiate the two disease outcomes, supporting model development and refinement for binary classification.

For that purpose, the authors used the `ITP2bspline()` function of the R package `{fdatest}` that implements the Interval Testing Procedure (ITP) to compare two functional populations (epidemic vs. non-epidemic) evaluated on a uniform grid. It represents data using B-spline basis functions and assesses the significance of each basis coefficient with interval-wise control of the family-wise error rate (FWER) (Pini & Vantini, 2016).

In this section, we first demonstrate how to fit the function-on-scalar regression with the objective of visualizing two curves, the difference coefficient function and the overall mean coefficient curve. The former shows how the effect of a weather-related variable on the response variable changes over the days relative to a reference date (e.g. planting date, flowering). It helps to identify critical windows when the weather variations have the most substantial impact. The latter is the mean weather curve of the reference group (non-epidemic studies); the difference coefficient is the difference between the mean curves of epidemic and non-epidemic studies.

Then, we demonstrate the Interval Testing Procedure (ITP) to compare two functional populations (epidemic vs. non-epidemic) and identify exactly the days in the series that the two populations are significantly different. We will keep analyzing the data on wheat blast epidemics in Brazil.

Function-on-scalar regression

The following analysis uses function-on-scalar regression to explore the relationship between relative humidity at 2 meters (RH2M) and the occurrence of wheat blast epidemics. Let's first load the necessary libraries.

```
library(dplyr)
library(tidyr)
library(r4pde)
library(refund)
library(ggplot2)
```

The first step involves reading and merging two datasets:

- `trials`: Contains disease data with heading dates.
- `weather_data`: Contains weather data for specific time windows. It was created in the previous section on window-pane and can be downloaded [here](#).

The datasets are merged into `trials_weather`, and the resulting data is transformed into a wide format where each column represents time points (days), and rows correspond to individual studies. The binary epidemic status is created, where:

- 1 indicates an epidemic (if `inc_mean > 20`).
- 0 indicates no epidemic (if `inc_mean <= 20`).

The weather variable (`RH2M`) is used as the predictor, and time points (days) are calculated relative to the heading date.

```
# load the disease dataset
trials <- r4pde::BlastWheat
trials$heading = as.Date(trials$heading, format = "%d-%m-%Y")

# load the weather data constructed in previous section
weather_data <- readr::read_csv("https://raw.githubusercontent.com/emdelponete/epidemiology-R/refs/heads/main/
data/weather_windowpane.csv")
trials_weather <- full_join(trials, weather_data)

# create epidemic variable and number of days relative to heading
dat_wide_RH <- trials_weather |>
  mutate(epidemic = if_else(inc_mean > 20, 1, 0),
         days = as.numeric(-(heading - YYYYMMDD))) |>
  dplyr::select(study, epidemic, days, RH2M) |>
  pivot_wider(id_cols = c(study, epidemic),
              names_from = days, values_from = RH2M)
```

The scalar covariate is the epidemic status (0 or 1), while the functional response is a matrix with the values of the specified weather variable across time (one row per study, one column per day). The model fits smooth functions to represent the effects of time and the epidemic status on the weather variable using penalized splines. The function-on-scalar regression model is given by:

$$y_{ij} = \beta_0(t_j) + \beta_1(t_j)x_i + \epsilon_{ij}$$

where: y_{ij} is the value of the weather variable for study i at time j ; $\beta_0(t_j)$ is the smooth function for the overall mean effect over time; $\beta_1(t_j)$ is the smooth function representing the effect of epidemic status over time; x_i is the binary epidemic status (0 or 1); and ϵ_{ij} is the error term.

We can now fit the model using the `pffr()` function of the `{refund}` package.

```
x <- dat_wide_RH$epidemic # Scalar covariate (epidemic status)
y <- dat_wide_RH |>
  dplyr::select(-study, -epidemic) |>
  as.matrix() # Functional response (weather variable by day)

yind <- as.numeric(colnames(y))

FOSR_RH <- pffr(y ~ x,
               yind = yind,
               bs.yindex = list(bs = "ps", k = 30, m = c(2, 1)),
               bs.int = list(bs = "ps", k = 30, m = c(2, 1)),
               algorithm = "gam")
```

Let's extract the coefficients of the model fit object and prepare the data and plot using `ggplot`.

```
coef_list <- coef(FOSR_RH)$smterms
```

```
using seWithMean for s(yind.vec) .
```

```
mean_df <- data.frame(
  time = coef_list[[1]]$coef[, "yind.vec"],
  value = coef_list[[1]]$coef[, "value"],
  lower = coef_list[[1]]$coef[, "value"] - 1.96 * coef_list[[1]]$coef[, "se"],
  upper = coef_list[[1]]$coef[, "value"] + 1.96 * coef_list[[1]]$coef[, "se"],
  term = "Overall Mean"
)

x_df <- data.frame(
  time = coef_list[[2]]$coef[, "yind.vec"],
  value = coef_list[[2]]$coef[, "value"],
  lower = coef_list[[2]]$coef[, "value"] - 1.96 * coef_list[[2]]$coef[, "se"],
  upper = coef_list[[2]]$coef[, "value"] + 1.96 * coef_list[[2]]$coef[, "se"],
  term = "Difference"
)

plot_df <- bind_rows(mean_df, x_df)

ggplot(plot_df, aes(x = time, y = value)) +
  geom_ribbon(aes(ymin = lower, ymax = upper),
    alpha = 0.2) +
  geom_line(linewidth = 1.2) +
  facet_grid(rows = vars(term),
    scales = "free_y") +
  theme_r4pde(font_size = 12) +
  geom_hline(aes(yintercept = 0),
    color = "grey",
    linetype = "dashed") +
  geom_vline(aes(xintercept = 0),
    color = "grey",
    linetype = "dashed") +
  labs(x = "Day relative to wheat heading", y = "Coefficient Function",
    title = "RH2M") +
  theme(strip.text = element_text(face = "bold", size = rel(1.0)),
    axis.title = element_text(face = "bold", size = 12))
```

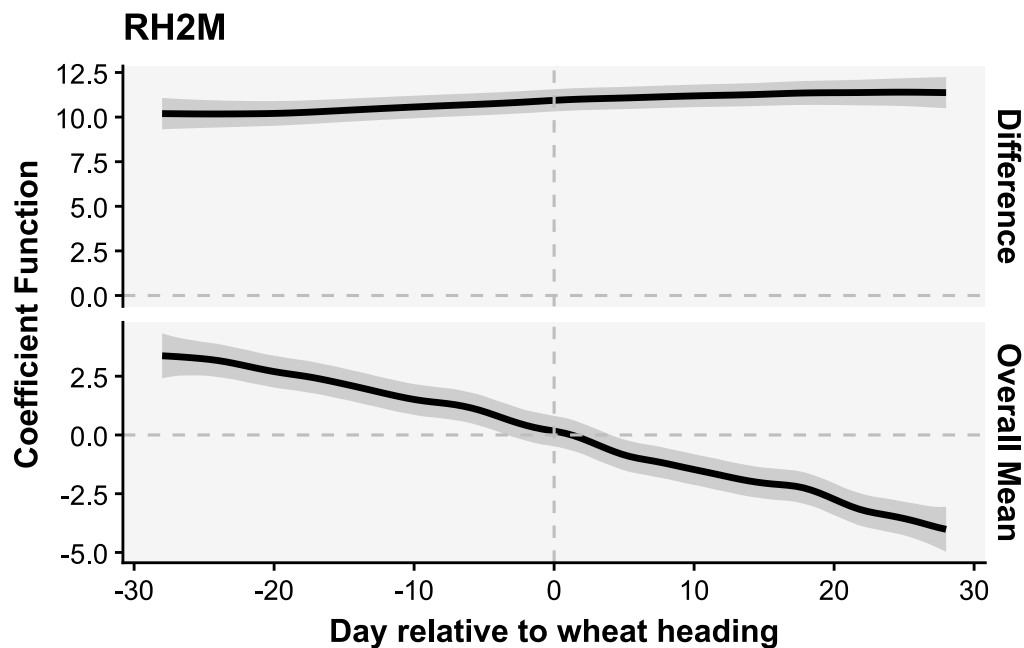


Figure 21.17: Function-on-scalar regression for relative humidity: overall mean curve and difference between epidemic and non-epidemic studies, with 95% confidence bands

Because we have several weather variables, we can facilitate the process by creating a function named `fosr()` that can be applied for the other variables.

```
# Define the function for function-on-scalar regression
fosr <- function(data, weather_var) {

  # Step 1: Prepare the dataset and transform to wide format
  dat_wide <- data |>
    mutate(epidemic = if_else(inc_mean > 20, 1, 0),
           days = as.numeric(-(heading - YYYYMMDD))) |>
    dplyr::select(study, epidemic, days, !!sym(weather_var)) |>
    pivot_wider(id_cols = c(study, epidemic), names_from = days, values_from = !!sym(weather_var))

  x <- dat_wide$epidemic # Scalar covariate (epidemic status)
  y <- dat_wide |>
    dplyr::select(-study, -epidemic) |>
    as.matrix() # Functional response (weather variable by day)

  m2 <- pffr(y ~ x,
             yind = yind,
             bs.yindex = list(bs = "ps", k = 30, m = c(2, 1)),
             bs.int = list(bs = "ps", k = 30, m = c(2, 1)),
             algorithm = "gam")

  coef_list <- coef(m2)$smterms

  mean_df <- data.frame(
    time = coef_list[[1]]$coef[, "yind.vec"],
    value = coef_list[[1]]$coef[, "value"],
    lower = coef_list[[1]]$coef[, "value"] - 1.96 * coef_list[[1]]$coef[, "se"],
    upper = coef_list[[1]]$coef[, "value"] + 1.96 * coef_list[[1]]$coef[, "se"],
```

```

    term = "Overall Mean"
  )

  x_df <- data.frame(
    time = coef_list[[2]]$coef[, "yind.vec"],
    value = coef_list[[2]]$coef[, "value"],
    lower = coef_list[[2]]$coef[, "value"] - 1.96 * coef_list[[2]]$coef[, "se"],
    upper = coef_list[[2]]$coef[, "value"] + 1.96 * coef_list[[2]]$coef[, "se"],
    term = "Difference"
  )

  plot_df <- bind_rows(mean_df, x_df)

  ggplot(plot_df, aes(x = time, y = value)) +
    geom_ribbon(aes(ymin = lower, ymax = upper),
              alpha = 0.2) +
    geom_line(linewidth = 1.2) +
    facet_grid(rows = vars(term),
              scales = "free_y") +
    theme_r4pde(font_size = 10) +
    geom_hline(aes(yintercept = 0),
              color = "grey",
              linetype = "dashed") +
    geom_vline(aes(xintercept = 0),
              color = "grey",
              linetype = "dashed") +
    labs(x = "Day relative to wheat heading", y = "Coefficient Function",
         title = weather_var) +
    theme(strip.text = element_text(face = "bold", size = rel(1.0)),
          axis.title = element_text(face = "bold", size = 10))
}

```

Now we can apply the function to the other variables and obtain plots for each variable.

```
tmin <- fosr(trials_weather, "T2M_MIN")
```

```
using seWithMean for s(yind.vec) .
```

```
tmax <- fosr(trials_weather, "T2M_MAX")
```

```
using seWithMean for s(yind.vec) .
```

```
rh <- fosr(trials_weather, "RH2M")
```

```
using seWithMean for s(yind.vec) .
```

```
prec <- fosr(trials_weather, "PRECTOTCORR")
```

```
using seWithMean for s(yind.vec) .
```

```
library(patchwork)

(tmin | tmax)/(rh | prec)
```

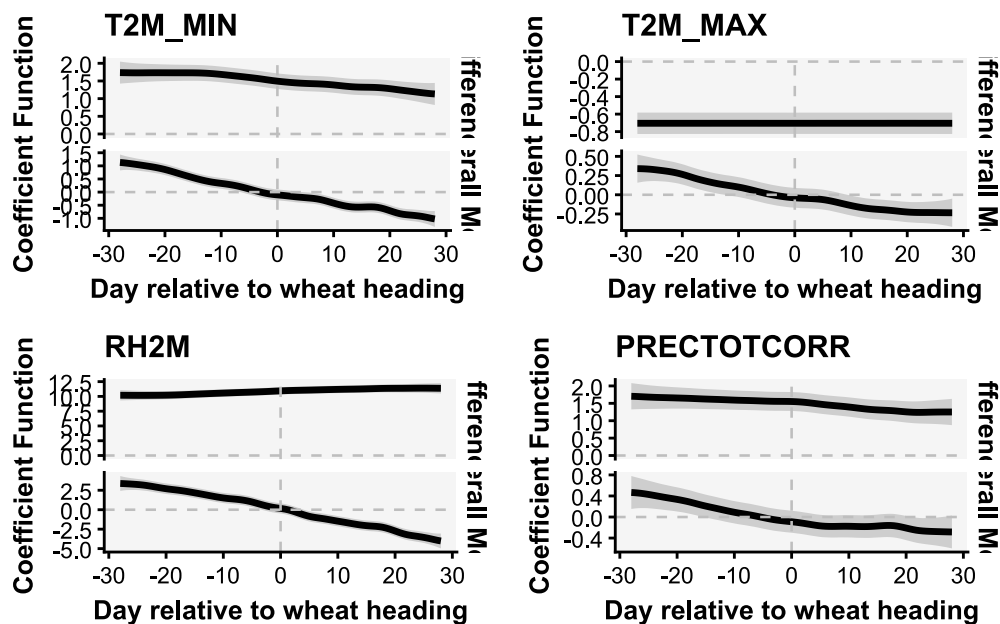


Figure 21.18: Function-on-scalar regression coefficient functions for minimum temperature, maximum temperature, relative humidity and precipitation

Interval test procedure

For this test, we need to subset the data into two epidemic conditions, transform from long to wide format and create a matrix object.

```
df_epidemic <- trials_weather |>
  mutate(epidemic = if_else(inc_mean > 20, 1, 0),
         days = as.numeric(-(heading - YYYYMMDD))) |>
  filter(epidemic == 1) |>
  dplyr::select(RH2M, days, study)

df_non_epidemic <- trials_weather |>
  mutate(epidemic = if_else(inc_mean > 20, 1, 0),
         days = as.numeric(-(heading - YYYYMMDD))) |>
  filter(epidemic == 0) |>
  dplyr::select(RH2M, days, study)

# Pivot data to wide format for FDA
df_epidemic_wide <- df_epidemic |>
  group_by(study) |>
  pivot_wider(names_from = days, values_from = RH2M) |>
  ungroup() |>
  dplyr::select(-study)

df_non_epidemic_wide <- df_non_epidemic |>
  group_by(study) |>
  pivot_wider(names_from = days, values_from = RH2M) |>
  ungroup() |>
```

```
dplyr::select(-study)

# Convert to matrix
data_epidemic <- as.matrix(df_epidemic_wide)
data_non_epidemic <- as.matrix(df_non_epidemic_wide)
```

Now we can apply the function specifying each new data in each data argument.

```
library(fdatest)

# Perform FDA test
itp_result <- ITP2bspline(data1 = data_epidemic,
                          data2 = data_non_epidemic,
                          B = 100)
```

```
[1] "First step: basis expansion"
Swapping 'y' and 'argvals', because 'y' is simpler,
and 'argvals' should be; now dim(argvals) = 57 ; dim(y) = 57 x 143
[1] "Second step: joint univariate tests"
[1] "Third step: interval-wise combination and correction"
[1] "creating the p-value matrix: end of row 2 out of 57"
[1] "creating the p-value matrix: end of row 3 out of 57"
[1] "creating the p-value matrix: end of row 4 out of 57"
[1] "creating the p-value matrix: end of row 5 out of 57"
[1] "creating the p-value matrix: end of row 6 out of 57"
[1] "creating the p-value matrix: end of row 7 out of 57"
[1] "creating the p-value matrix: end of row 8 out of 57"
[1] "creating the p-value matrix: end of row 9 out of 57"
[1] "creating the p-value matrix: end of row 10 out of 57"
[1] "creating the p-value matrix: end of row 11 out of 57"
[1] "creating the p-value matrix: end of row 12 out of 57"
[1] "creating the p-value matrix: end of row 13 out of 57"
[1] "creating the p-value matrix: end of row 14 out of 57"
[1] "creating the p-value matrix: end of row 15 out of 57"
[1] "creating the p-value matrix: end of row 16 out of 57"
[1] "creating the p-value matrix: end of row 17 out of 57"
[1] "creating the p-value matrix: end of row 18 out of 57"
[1] "creating the p-value matrix: end of row 19 out of 57"
[1] "creating the p-value matrix: end of row 20 out of 57"
[1] "creating the p-value matrix: end of row 21 out of 57"
[1] "creating the p-value matrix: end of row 22 out of 57"
[1] "creating the p-value matrix: end of row 23 out of 57"
[1] "creating the p-value matrix: end of row 24 out of 57"
[1] "creating the p-value matrix: end of row 25 out of 57"
[1] "creating the p-value matrix: end of row 26 out of 57"
[1] "creating the p-value matrix: end of row 27 out of 57"
[1] "creating the p-value matrix: end of row 28 out of 57"
[1] "creating the p-value matrix: end of row 29 out of 57"
[1] "creating the p-value matrix: end of row 30 out of 57"
[1] "creating the p-value matrix: end of row 31 out of 57"
[1] "creating the p-value matrix: end of row 32 out of 57"
[1] "creating the p-value matrix: end of row 33 out of 57"
[1] "creating the p-value matrix: end of row 34 out of 57"
[1] "creating the p-value matrix: end of row 35 out of 57"
[1] "creating the p-value matrix: end of row 36 out of 57"
[1] "creating the p-value matrix: end of row 37 out of 57"
[1] "creating the p-value matrix: end of row 38 out of 57"
```

```
[1] "creating the p-value matrix: end of row 39 out of 57"
[1] "creating the p-value matrix: end of row 40 out of 57"
[1] "creating the p-value matrix: end of row 41 out of 57"
[1] "creating the p-value matrix: end of row 42 out of 57"
[1] "creating the p-value matrix: end of row 43 out of 57"
[1] "creating the p-value matrix: end of row 44 out of 57"
[1] "creating the p-value matrix: end of row 45 out of 57"
[1] "creating the p-value matrix: end of row 46 out of 57"
[1] "creating the p-value matrix: end of row 47 out of 57"
[1] "creating the p-value matrix: end of row 48 out of 57"
[1] "creating the p-value matrix: end of row 49 out of 57"
[1] "creating the p-value matrix: end of row 50 out of 57"
[1] "creating the p-value matrix: end of row 51 out of 57"
[1] "creating the p-value matrix: end of row 52 out of 57"
[1] "creating the p-value matrix: end of row 53 out of 57"
[1] "creating the p-value matrix: end of row 54 out of 57"
[1] "creating the p-value matrix: end of row 55 out of 57"
[1] "creating the p-value matrix: end of row 56 out of 57"
[1] "creating the p-value matrix: end of row 57 out of 57"
[1] "Interval Testing Procedure completed"
```

Here, we can see the p-values for each time point as well as which time points were significant considering $p < 0.05$.

```
# global p-values
itp_result$corrected.pval
```

[illegible]

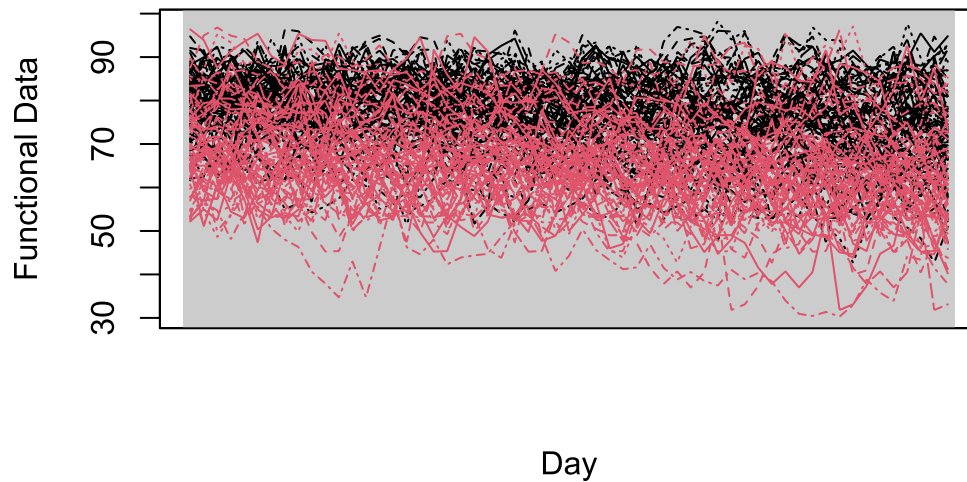
```
# significant components
which(itp_result$corrected.pval < 0.05)
```

```
[1]  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] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
[51] 51 52 53 54 55 56 57
```

The plot function produces two graphs, one for the curves and another for the p-values where the shaded area indicates the significance.

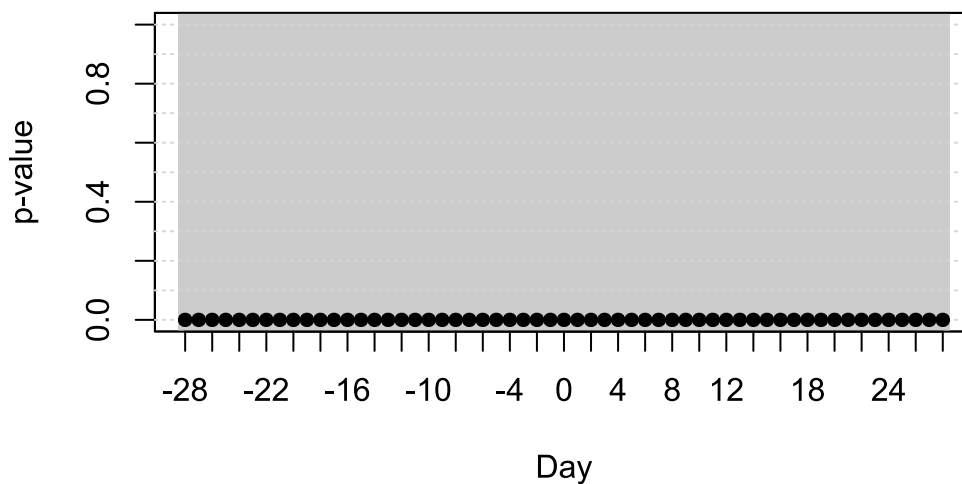
```
# Plot FDA results
plot(itp_result, main = "RH2M",
     xrange = c(-28, 28),
     xlab = 'Day', xaxt = 'n')
```

RH2M : Functional Data



```
axis(1, at = seq(-28, 28, by = 2),
     labels = seq(-28, 28, by = 2))
```

RH2M : Adjusted p-values



Because we may have several weather variables, we can avoid repeating the code for each variable by creating a function. The arguments are the data and weather variable. Optionally, the user can define the threshold for disease.

```
run_ITP_test <- function(data, weather_var, threshold = 20, B = 100) {
  # Convert the weather_var input to symbol for dplyr's select function
  weather_var_sym <- sym(weather_var)

  # Filter and prepare epidemic data
```



```

df_epidemic <- data |>
  mutate(epidemic = if_else(inc_mean > threshold, 1, 0),
         days = as.numeric(-(heading - YYYYMMDD))) |>
  filter(epidemic == 1) |>
  dplyr::select(!weather_var_sym, days, study)

# Filter and prepare non-epidemic data
df_non_epidemic <- data |>
  mutate(epidemic = if_else(inc_mean > threshold, 1, 0),
         days = as.numeric(-(heading - YYYYMMDD))) |>
  filter(epidemic == 0) |>
  dplyr::select(!weather_var_sym, days, study)

# Pivot epidemic data to wide format
df_epidemic_wide <- df_epidemic |>
  group_by(study) |>
  pivot_wider(names_from = days, values_from = !!weather_var_sym) |>
  ungroup() |>
  dplyr::select(-study)

# Pivot non-epidemic data to wide format
df_non_epidemic_wide <- df_non_epidemic |>
  group_by(study) |>
  pivot_wider(names_from = days, values_from = !!weather_var_sym) |>
  ungroup() |>
  dplyr::select(-study)

# Convert to matrix
data_epidemic <- as.matrix(df_epidemic_wide)
data_non_epidemic <- as.matrix(df_non_epidemic_wide)

# Perform FDA test
itp_result <- ITP2bspline(data1 = data_epidemic,
                          data2 = data_non_epidemic,
                          B = B)

return(itp_result)
}

```

Applying the function to each variable.

```
itp_tmin <- run_ITP_test(data = trials_weather, weather_var = "T2M_MIN")
```

```

[1] "First step: basis expansion"
Swapping 'y' and 'argvals', because 'y' is simpler,
and 'argvals' should be; now dim(argvals) = 57 ; dim(y) = 57 x 143
[1] "Second step: joint univariate tests"
[1] "Third step: interval-wise combination and correction"
[1] "creating the p-value matrix: end of row 2 out of 57"
[1] "creating the p-value matrix: end of row 3 out of 57"
[1] "creating the p-value matrix: end of row 4 out of 57"
[1] "creating the p-value matrix: end of row 5 out of 57"
[1] "creating the p-value matrix: end of row 6 out of 57"
[1] "creating the p-value matrix: end of row 7 out of 57"
[1] "creating the p-value matrix: end of row 8 out of 57"
[1] "creating the p-value matrix: end of row 9 out of 57"
[1] "creating the p-value matrix: end of row 10 out of 57"

```

```

[1] "creating the p-value matrix: end of row 11 out of 57"
[1] "creating the p-value matrix: end of row 12 out of 57"
[1] "creating the p-value matrix: end of row 13 out of 57"
[1] "creating the p-value matrix: end of row 14 out of 57"
[1] "creating the p-value matrix: end of row 15 out of 57"
[1] "creating the p-value matrix: end of row 16 out of 57"
[1] "creating the p-value matrix: end of row 17 out of 57"
[1] "creating the p-value matrix: end of row 18 out of 57"
[1] "creating the p-value matrix: end of row 19 out of 57"
[1] "creating the p-value matrix: end of row 20 out of 57"
[1] "creating the p-value matrix: end of row 21 out of 57"
[1] "creating the p-value matrix: end of row 22 out of 57"
[1] "creating the p-value matrix: end of row 23 out of 57"
[1] "creating the p-value matrix: end of row 24 out of 57"
[1] "creating the p-value matrix: end of row 25 out of 57"
[1] "creating the p-value matrix: end of row 26 out of 57"
[1] "creating the p-value matrix: end of row 27 out of 57"
[1] "creating the p-value matrix: end of row 28 out of 57"
[1] "creating the p-value matrix: end of row 29 out of 57"
[1] "creating the p-value matrix: end of row 30 out of 57"
[1] "creating the p-value matrix: end of row 31 out of 57"
[1] "creating the p-value matrix: end of row 32 out of 57"
[1] "creating the p-value matrix: end of row 33 out of 57"
[1] "creating the p-value matrix: end of row 34 out of 57"
[1] "creating the p-value matrix: end of row 35 out of 57"
[1] "creating the p-value matrix: end of row 36 out of 57"
[1] "creating the p-value matrix: end of row 37 out of 57"
[1] "creating the p-value matrix: end of row 38 out of 57"
[1] "creating the p-value matrix: end of row 39 out of 57"
[1] "creating the p-value matrix: end of row 40 out of 57"
[1] "creating the p-value matrix: end of row 41 out of 57"
[1] "creating the p-value matrix: end of row 42 out of 57"
[1] "creating the p-value matrix: end of row 43 out of 57"
[1] "creating the p-value matrix: end of row 44 out of 57"
[1] "creating the p-value matrix: end of row 45 out of 57"
[1] "creating the p-value matrix: end of row 46 out of 57"
[1] "creating the p-value matrix: end of row 47 out of 57"
[1] "creating the p-value matrix: end of row 48 out of 57"
[1] "creating the p-value matrix: end of row 49 out of 57"
[1] "creating the p-value matrix: end of row 50 out of 57"
[1] "creating the p-value matrix: end of row 51 out of 57"
[1] "creating the p-value matrix: end of row 52 out of 57"
[1] "creating the p-value matrix: end of row 53 out of 57"
[1] "creating the p-value matrix: end of row 54 out of 57"
[1] "creating the p-value matrix: end of row 55 out of 57"
[1] "creating the p-value matrix: end of row 56 out of 57"
[1] "creating the p-value matrix: end of row 57 out of 57"
[1] "Interval Testing Procedure completed"

```

```
itp_tmax <- run_ITP_test(data = trials_weather, weather_var = "T2M_MAX")
```

```

[1] "First step: basis expansion"
Swapping 'y' and 'argvals', because 'y' is simpler,
and 'argvals' should be; now dim(argvals) = 57 ; dim(y) = 57 x 143
[1] "Second step: joint univariate tests"
[1] "Third step: interval-wise combination and correction"

```

413

```
itp_prec <- run_ITP_test(data = trials_weather, weather_var = "PRECTOTCORR")
```

```
[1] "First step: basis expansion"
Swapping 'y' and 'argvals', because 'y' is simpler,
  and 'argvals' should be; now dim(argvals) = 57 ; dim(y) = 57 x 143
[1] "Second step: joint univariate tests"
[1] "Third step: interval-wise combination and correction"
[1] "creating the p-value matrix: end of row 2 out of 57"
[1] "creating the p-value matrix: end of row 3 out of 57"
[1] "creating the p-value matrix: end of row 4 out of 57"
[1] "creating the p-value matrix: end of row 5 out of 57"
[1] "creating the p-value matrix: end of row 6 out of 57"
[1] "creating the p-value matrix: end of row 7 out of 57"
[1] "creating the p-value matrix: end of row 8 out of 57"
[1] "creating the p-value matrix: end of row 9 out of 57"
[1] "creating the p-value matrix: end of row 10 out of 57"
[1] "creating the p-value matrix: end of row 11 out of 57"
[1] "creating the p-value matrix: end of row 12 out of 57"
[1] "creating the p-value matrix: end of row 13 out of 57"
[1] "creating the p-value matrix: end of row 14 out of 57"
[1] "creating the p-value matrix: end of row 15 out of 57"
[1] "creating the p-value matrix: end of row 16 out of 57"
[1] "creating the p-value matrix: end of row 17 out of 57"
[1] "creating the p-value matrix: end of row 18 out of 57"
[1] "creating the p-value matrix: end of row 19 out of 57"
[1] "creating the p-value matrix: end of row 20 out of 57"
[1] "creating the p-value matrix: end of row 21 out of 57"
[1] "creating the p-value matrix: end of row 22 out of 57"
[1] "creating the p-value matrix: end of row 23 out of 57"
[1] "creating the p-value matrix: end of row 24 out of 57"
[1] "creating the p-value matrix: end of row 25 out of 57"
[1] "creating the p-value matrix: end of row 26 out of 57"
[1] "creating the p-value matrix: end of row 27 out of 57"
[1] "creating the p-value matrix: end of row 28 out of 57"
[1] "creating the p-value matrix: end of row 29 out of 57"
[1] "creating the p-value matrix: end of row 30 out of 57"
[1] "creating the p-value matrix: end of row 31 out of 57"
[1] "creating the p-value matrix: end of row 32 out of 57"
[1] "creating the p-value matrix: end of row 33 out of 57"
[1] "creating the p-value matrix: end of row 34 out of 57"
[1] "creating the p-value matrix: end of row 35 out of 57"
[1] "creating the p-value matrix: end of row 36 out of 57"
[1] "creating the p-value matrix: end of row 37 out of 57"
[1] "creating the p-value matrix: end of row 38 out of 57"
[1] "creating the p-value matrix: end of row 39 out of 57"
[1] "creating the p-value matrix: end of row 40 out of 57"
[1] "creating the p-value matrix: end of row 41 out of 57"
[1] "creating the p-value matrix: end of row 42 out of 57"
[1] "creating the p-value matrix: end of row 43 out of 57"
[1] "creating the p-value matrix: end of row 44 out of 57"
[1] "creating the p-value matrix: end of row 45 out of 57"
[1] "creating the p-value matrix: end of row 46 out of 57"
[1] "creating the p-value matrix: end of row 47 out of 57"
[1] "creating the p-value matrix: end of row 48 out of 57"
[1] "creating the p-value matrix: end of row 49 out of 57"
[1] "creating the p-value matrix: end of row 50 out of 57"
[1] "creating the p-value matrix: end of row 51 out of 57"
```

```
[1] "creating the p-value matrix: end of row 52 out of 57"
[1] "creating the p-value matrix: end of row 53 out of 57"
[1] "creating the p-value matrix: end of row 54 out of 57"
[1] "creating the p-value matrix: end of row 55 out of 57"
[1] "creating the p-value matrix: end of row 56 out of 57"
[1] "creating the p-value matrix: end of row 57 out of 57"
[1] "Interval Testing Procedure completed"
```

```
itp_tmin$corrected.pval
```

```
[1] 0.00 0.00 0.04 0.00 0.00 0.00 0.00 0.00 0.02 0.00 0.00 0.00 0.00 0.00
[16] 0.00 0.00 0.00 0.01 0.01 0.01 0.01 0.01 0.01 0.00 0.00 0.00 0.06 0.07 0.02
[31] 0.02 0.03 0.02 0.03 0.00 0.00 0.00 0.01 0.00 0.03 0.04 0.05 0.19 0.08 0.03
[46] 0.00 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.11 0.08 0.15
```

```
which(itp_tmin$corrected.pval < 0.05)
```

```
[1] 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] 26 27 30 31 32 33 34 35 36 37 38 39 40 41 45 46 47 48 49 50 51 52 53 54
```

```
itp_tmax$corrected.pval
```

```
[1] 0.00 0.00 0.04 0.14 0.07 0.09 0.09 0.10 0.18 0.58 0.28 0.19 0.14 0.27 0.35
[16] 0.43 0.39 0.42 0.54 0.29 0.13 0.17 0.17 0.25 0.38 0.51 0.35 0.23 0.12 0.12
[31] 0.13 0.15 0.15 0.19 0.20 0.21 0.25 0.47 0.29 0.29 0.31 0.29 0.31 0.45 0.39
[46] 0.39 0.60 0.89 0.92 0.72 0.58 0.56 0.44 0.24 0.00 0.00 0.04
```

```
which(itp_tmax$corrected.pval < 0.05)
```

```
[1] 1 2 3 55 56 57
```

```
itp_prec$corrected.pval
```

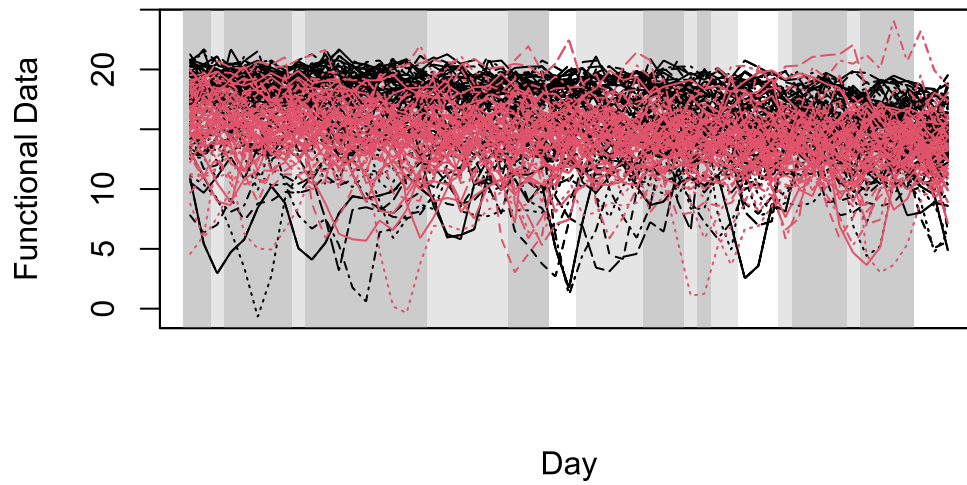
```
[1] 0.00 0.19 0.55 0.22 0.38 0.09 0.00 0.00 0.90 0.00 0.03 0.09 0.00 0.00 0.29
[16] 0.00 0.00 0.00 0.00 0.44 0.00 0.03 0.00 0.04 0.15 0.06 0.00 0.00 0.00 0.00
[31] 0.00 0.00 0.00 0.25 0.00 0.00 0.15 0.07 0.00 0.00 0.08 0.18 0.87 0.51 0.33
[46] 0.32 0.74 0.92 0.92 0.92 0.21 0.08 0.00 0.01 0.01 0.05 0.03
```

```
which(itp_prec$corrected.pval < 0.05)
```

```
[1] 1 7 8 10 11 13 14 16 17 18 19 21 22 23 24 27 28 29 30 31 32 33 35 36 39
[26] 40 53 54 55 57
```

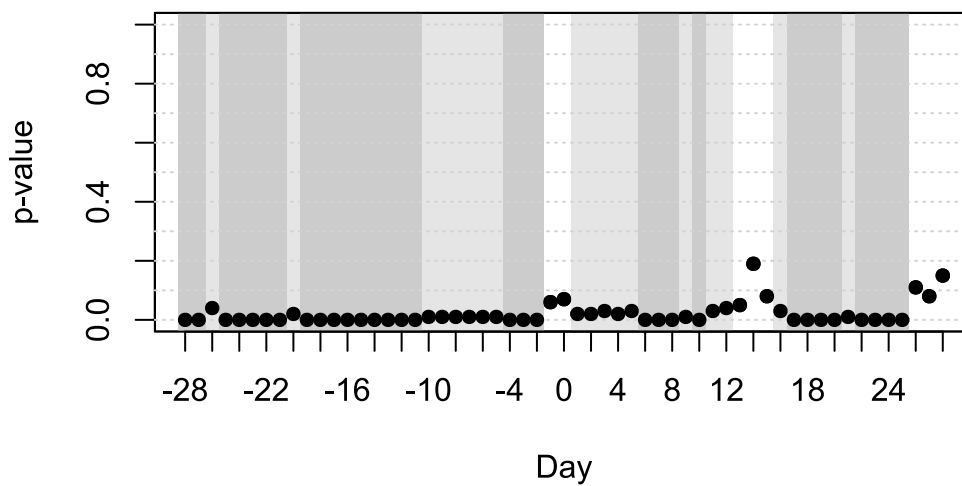
```
plot(itp_tmin, main = "TMIN",
     xrange = c(-28, 28),
     xlab = 'Day', xaxt = 'n')
```

TMIN : Functional Data



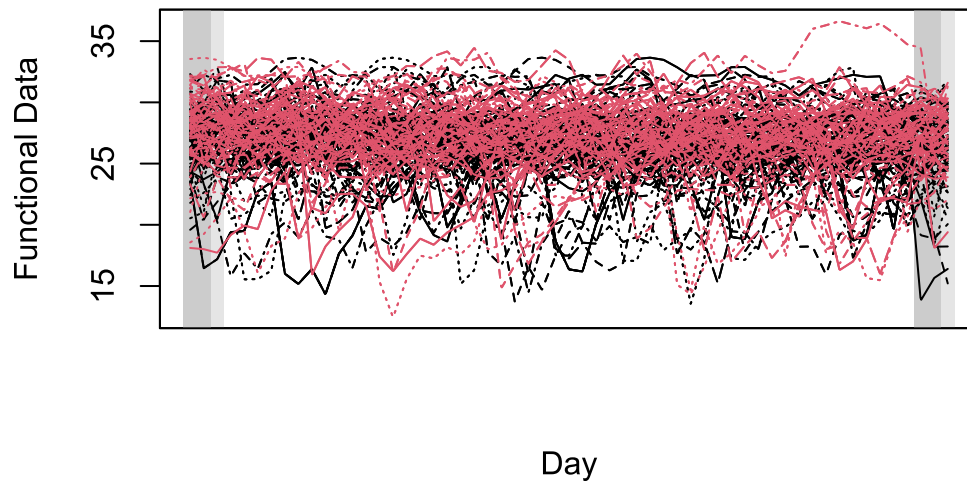
```
axis(1, at = seq(-28, 28, by = 2),
     labels = seq(-28, 28, by = 2))
```

TMIN : Adjusted p-values



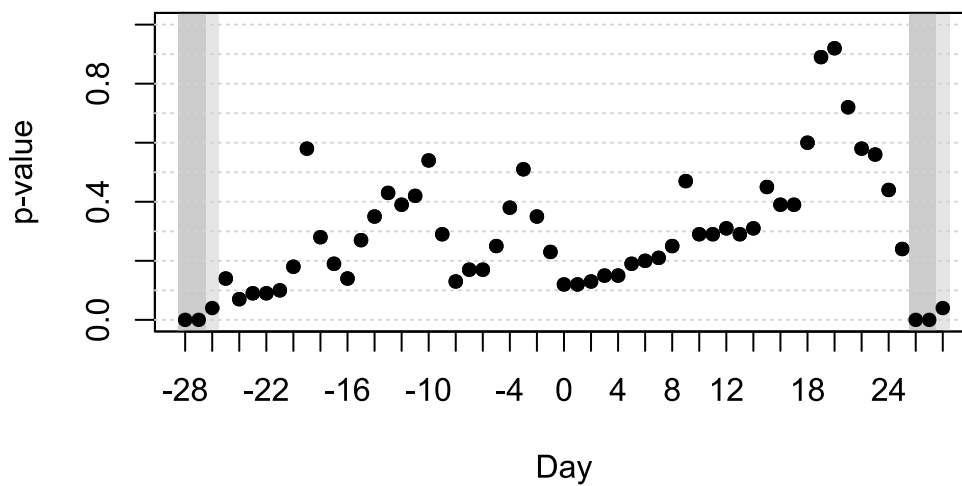
```
plot(itp_tmax, main = "TMAX",
     xrange = c(-28, 28),
     xlab = 'Day', xaxt = 'n')
```

TMAX : Functional Data



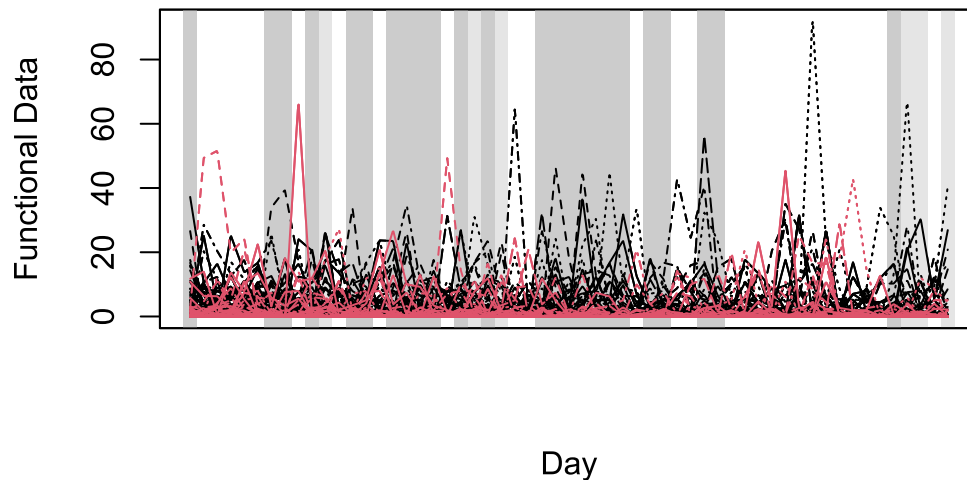
```
axis(1, at = seq(-28, 28, by = 2),
     labels = seq(-28, 28, by = 2))
```

TMAX : Adjusted p-values



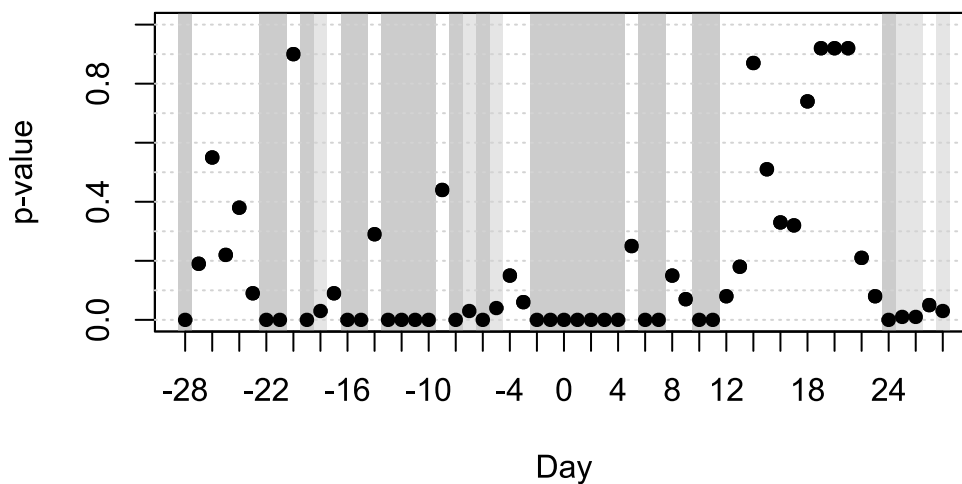
```
plot(itp_prec, main = "PREC",
     xrange = c(-28, 28),
     xlab = 'Day', xaxt = 'n')
```

PREC : Functional Data



```
axis(1, at = seq(-28, 28, by = 2),
     labels = seq(-28, 28, by = 2))
```

PREC : Adjusted p-values



Interpretation: Based on the results above, relative humidity differed significantly between epidemic and non-epidemic studies on every day of the series, and minimum temperature on most days, particularly during the pre-heading period. Maximum temperature differed only on a few isolated days. Lastly, precipitation influenced wheat blast during several short intervals, with the most notable impact occurring over a longer seven-day period around the heading stage. Therefore, TMIN, RH and PREC can be tested further as candidate predictors in prediction models.

Model fitting

Logistic regression models

Logistic regression is a statistical modeling technique used to predict the probability of a binary outcome based on one or more predictor variables (James et al., 2021). In the context of plant disease prediction, the logistic model relates factors like environmental conditions (usually weather), host characteristics (e.g., cultivar resistance), and pathogen presence to the likelihood of disease occurrence. This occurrence can be expressed as presence or absence (Mila et al., 2004) or as a classification of incidence/severity based on an epidemic threshold (De Cól et al., 2024; De Wolf et al., 2003). By integrating these variables, the model enables researchers to assess disease risk under varying conditions and to develop decision support systems. This modeling approach has been used successfully in several studies in the field (De Cól et al., 2024; De Wolf et al., 2003; Harikrishnan & Río, 2008; Mila et al., 2004; Romero et al., 2021; Shah et al., 2013; Willbur et al., 2018; Xu et al., 2013), demonstrating its efficacy in predicting disease outbreaks and informing early intervention efforts.

The logistic model uses the logistic function to transform a linear combination of predictors into a probability between 0 and 1. This transformation ensures that the predicted values are valid probabilities, facilitating meaningful interpretations. By estimating the influence of each predictor on the odds of disease, logistic regression helps identify significant risk factors. Additionally, the model's coefficients provide insights into the strength and direction of the association between predictors and disease risk, offering valuable information for understanding the underlying mechanisms of disease development and progression.

The logistic regression model predicts the probability P of the binary outcome $Y = 1 \mid X$ (e.g., presence or absence of disease) given a set of predictor variables X . The model uses the logistic function to ensure the predicted probabilities lie between 0 and 1:

$$P(Y = 1 \mid X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p)}}$$

Alternatively, the model can be expressed in terms of the log-odds (also known as the logit function):

$$\log\left(\frac{P(Y=1|X)}{1-P(Y=1|X)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

Compared to more complex machine learning models, logistic regression offers a high degree of interpretability, which is crucial in many scientific and practical applications. While models like neural networks, random forests, or support vector machines can achieve high predictive accuracy, they often function as "black boxes" where the relationship between predictors and the outcome is not easily understood (Molnar, 2019). In contrast, logistic regression provides clear and direct interpretations of model parameters, allowing researchers and practitioners to understand how each predictor variable influences the probability of disease occurrence. This transparency makes logistic regression a preferred choice in contexts where explaining the underlying relationships is as important as prediction accuracy. The balance between interpretability and predictive performance justifies the choice of logistic regression in several contexts, especially when insights into the data are essential for decision-making, which is the case in plant disease prediction contexts.

In this section, we will keep working with the wheat blast disease data and expand on the previous sections on variable construction and selection, a usual step prior to testing the prediction variables in the logistic regression model.

Data preparation

```
library(r4pde)
library(tidyverse)

# load the disease dataset
trials <- r4pde::BlastWheat
trials$heading = as.Date(trials$heading, format = "%d-%m-%Y")

# load the weather data
weather_data <- readr::read_csv("https://raw.githubusercontent.com/emdelponte/epidemiology-R/refs/heads/main/
data/weather_windowpane.csv")

# join the two datasets
trials_weather <- full_join(trials, weather_data)

# create epidemic variable and number of days relative to heading
trials_weather <- trials_weather |>
  mutate(epidemic = if_else(inc_mean > 20, 1, 0),
         days = as.numeric(-(heading - YYYYMMDD)))
```

Let's create a few weather-related variables based on the original variables available from the NASA POWER. First, we need to group by study and epidemic and then filter the days to the period from 28 days before to 7 days after heading, since we will attempt to develop a model using only variables from the period around heading. We will not include mean daily maximum temperature given its weak association based on window-pane and not significant influence based on the interval test procedure of the FDA (see previous section).

```
epi_weather <- trials_weather |>
  group_by(study, epidemic) |>
  filter(days >= -28 & days <= 7) |>
  summarise(
    tmin_pre = mean(T2M_MIN, na.rm = TRUE),
    rh_pre = mean(RH2M, na.rm = TRUE),
    prec_pre = sum(PRECTOTCORR, na.rm = TRUE)) |>
  ungroup() |>
  dplyr::select(- study)

## create rainfall around heading
epi_rain <- trials_weather |>
  group_by(study, epidemic) |>
  filter(days >= -3 & days <= 6) |>
  summarise(
    prec_pos = sum(PRECTOTCORR, na.rm = TRUE)) |>
  ungroup() |>
  dplyr::select(- study, - epidemic)

epi_weather2 <- cbind(epi_weather, epi_rain)
```

Model fitting

Now we can fit the model and look at the summary.

```
# model fitting
m_logistic <- glm(epidemic ~ tmin_pre + rh_pre + prec_pre,
                  data = epi_weather,
```

```

family = binomial)

# Model summary
summary(m_logistic)

Call:
glm(formula = epidemic ~ tmin_pre + rh_pre + prec_pre, family = binomial,
    data = epi_weather)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -18.965637   3.921175  -4.837 1.32e-06 ***
tmin_pre      0.274067   0.091101   3.008 0.00263 **
rh_pre        0.201025   0.047644   4.219 2.45e-05 ***
prec_pre      0.002652   0.007112   0.373 0.70924
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 197.06  on 142  degrees of freedom
Residual deviance: 121.55  on 139  degrees of freedom
AIC: 129.55

Number of Fisher Scoring iterations: 5

```

interpretation of the coefficients:

- Intercept: Estimate of -18.96, indicating the baseline log-odds of an epidemic when all predictors are zero.
- tmin_pre: Positive estimate (0.27, $p = 0.002$), indicating that higher minimum temperatures are significantly associated with an increased probability of an epidemic.
- rh_pre: Positive estimate (0.20, $p < 0.001$), suggesting that higher relative humidity is strongly associated with an increased probability of an epidemic.
- prec_pre: Positive but non-significant estimate (0.003, $p = 0.71$), indicating no significant relationship between precipitation and epidemic occurrence.

Model fit

- The null deviance (197.06) reflects the fit of a model without predictors, while the residual deviance (121.55) indicates the fit of the full model with predictors. A significant reduction in deviance suggests that the predictors improve model fit.
- The AIC (129.55) helps evaluate model performance, with lower values indicating better fit.

Let's fit again the model using only the two significant variables and include the prec_pos variable.

```

m_logistic <- glm(epidemic ~ tmin_pre + rh_pre + prec_pos,
                  data = epi_weather2,
                  family = binomial)

summary(m_logistic)

```

```
Call:
glm(formula = epidemic ~ tmin_pre + rh_pre + prec_pos, family = binomial,
    data = epi_weather2)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -19.31170    3.47903  -5.551 2.84e-08 ***
tmin_pre      0.34620    0.10023   3.454 0.000552 ***
rh_pre        0.18418    0.03746   4.917 8.80e-07 ***
prec_pos      0.06393    0.02222   2.878 0.004003 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 197.06  on 142  degrees of freedom
Residual deviance: 109.99  on 139  degrees of freedom
AIC: 117.99

Number of Fisher Scoring iterations: 6
```

We can notice that including rainfall accumulation around flowering, the AIC is reduced to 117.99, suggesting that its inclusion improved prediction.

We can check the accuracy of the new model in predicting the training data.

```
library(caret)

# Obtain the predicted probability
pred_probs <- predict(m_logistic, epi_weather2, type = "response")

# Convert probabilities to binary outcomes using 0.5 as the threshold
pred_classes <- ifelse(pred_probs >= 0.5, 1, 0)

# Calculate accuracy
actual_classes <- epi_weather$epidemic
accuracy <- mean(pred_classes == actual_classes)
accuracy
```

```
[1] 0.8181818
```

The `confusionMatrix()` function of the `{caret}` package gives a more complete information about the performance of the model on the training set.

```
# create the confusion matrix
confusionMatrix(data = as.factor(as.numeric(pred_probs > 0.50)), mode = "everything", reference =
as.factor(epi_weather$epidemic),
    positive = "1")
```

Confusion Matrix and Statistics

```
          Reference
Prediction 0  1
0    49 10
1    16 68
```

```

        Accuracy : 0.8182
        95% CI : (0.7451, 0.8777)
    No Information Rate : 0.5455
    P-Value [Acc > NIR] : 6.459e-12

        Kappa : 0.6305

    Mcnemar's Test P-Value : 0.3268

        Sensitivity : 0.8718
        Specificity : 0.7538
    Pos Pred Value : 0.8095
    Neg Pred Value : 0.8305
        Precision : 0.8095
        Recall : 0.8718
         F1 : 0.8395
        Prevalence : 0.5455
    Detection Rate : 0.4755
    Detection Prevalence : 0.5874
    Balanced Accuracy : 0.8128

    'Positive' Class : 1

```

Model evaluation

It is highly desirable to evaluate model performance metrics using data that was not “seen” by the model during training. If the dataset is sufficiently large (e.g., more than 300 observations), it can be split into training and test sets, typically using an 80:20 ratio. For smaller datasets, alternatives like leave-one-out cross-validation (LOOCV) can be applied, as was done in the original wheat blast model study, which had 143 cases available (De Cól et al., 2024).

The Leave-One-Out Cross-Validation (LOOCV) is a resampling method used to evaluate model performance, particularly with small datasets. In LOOCV, the model is trained repeatedly, each time using all but one observation, which is set aside for testing. This process is repeated until every observation has served as the test case once. The overall performance metric is calculated by averaging the results from all iterations, providing an unbiased estimate of model accuracy (Hastie et al., 2009).

Let’s first create the training and test data.

```

train_index <- createDataPartition(epi_weather2$epidemic, p = 0.8, list = FALSE)
train_data <- epi_weather2[train_index, ]
test_data <- epi_weather2[-train_index, ]

```

Now we fit the logistic model on the training data and obtain the predictions.

```

# fit the model on training data
m_logistic2 <- glm(epidemic ~ tmin_pre + rh_pre + prec_pos,
                  data = train_data,
                  family = binomial)

# Predict on the test data
pred <- predict(m_logistic2, newdata = test_data, type = "response")

```

Let’s build now the confusion matrix for the test data.

```
confusion_matrix <- confusionMatrix(data = as.factor(as.numeric(pred > 0.5)), reference =
as.factor(test_data$epidemic), mode = "everything",
                                   positive = "1")
print(confusion_matrix)
```

Confusion Matrix and Statistics

```

      Reference
Prediction 0  1
      0  7  2
      1  5 14

      Accuracy : 0.75
      95% CI : (0.5513, 0.8931)
      No Information Rate : 0.5714
      P-Value [Acc > NIR] : 0.04018

      Kappa : 0.4731

      McNemar's Test P-Value : 0.44969

      Sensitivity : 0.8750
      Specificity : 0.5833
      Pos Pred Value : 0.7368
      Neg Pred Value : 0.7778
      Precision : 0.7368
      Recall : 0.8750
      F1 : 0.8000
      Prevalence : 0.5714
      Detection Rate : 0.5000
      Detection Prevalence : 0.6786
      Balanced Accuracy : 0.7292

      'Positive' Class : 1
```

The code below provides a comparative evaluation of model accuracy using two different cross-validation techniques, the LOOCV and 10-fold cross validation. The latter is similar to the LOOCV code, but with the addition of `k = 10`, which specifies 10-fold cross-validation. Here, the dataset is divided into 10 subsets (folds), with the model trained on 9 folds and tested on the remaining fold. This process repeats for all 10 folds, averaging the results. For both we need a custom cost function that evaluates the performance of a binary classifier. Note that we will use the model 1 built with all observations, not the training data.

```
library(boot)

cost <- function(r, pi = 0) mean(abs(r - pi) > 0.5)

# LOOCV (accuracy)
1 - cv.glm(epi_weather2, m_logistic, cost = cost)$delta[1]
```

```
[1] 0.8041958
```

```
# K-fold CV K=10 (accuracy)
1 - cv.glm(eps_weather2, m_logistic, K = 10, cost = cost)$delta[1]
```

```
[1] 0.7972028
```

We can conclude that using two variables from relatively long periods before heading, and the precipitation accumulation variable around heading (all defined based on FDA analysis) the model can reasonably predict wheat blast. We can further explore the effect of other variables from the post-heading period, as well as variables from shorter intervals around heading and flowering—periods when infections are likely to occur.

Variable selection is a critical process that combines statistical techniques with biological knowledge to identify the most relevant predictors for a given model. Approaches can range from simple methods, like selecting the most correlated variables at periods of time known as most critical for infection or epidemic development (De Wolf et al., 2003), to more advanced algorithms designed to optimize predictive power, such as regularization techniques like lasso or elastic-net (Dalla Lana et al., 2021b; De Cól et al., 2024; De Wolf et al., 2023).

Regularization methods not only help in selecting significant predictors but also in handling multicollinearity and reducing model complexity (Hastie et al., 2009).

References

- Adrakey, H., Streftaris, G., Cuniffe, N., Gottwald, T., Gilligan, C., & Gibson, G. (2017). Controlling spatio-temporal epidemics using latent processes in a Bayesian framework. *Journal of the Royal Society: Interface*, 14, 20170386.
- Agrios, G. N. (2005a). *INTRODUCTION* (pp. 3–75). Elsevier. <https://doi.org/10.1016/b978-0-08-047378-9.50007-5>
- Agrios, G. N. (2005b). *Plant Disease Epidemiology* (pp. 265–291). Elsevier. <https://doi.org/10.1016/b978-0-08-047378-9.50014-2>
- Allen, L., Bokil, V., Cuniffe, N., Hamelin, F., Hilker, F., & Jeger, M. (2019). Modelling vector transmission and epidemiology of co-infecting plant viruses. *Viruses*, 11, 1153.
- Alves, K. S., & Del Ponte, E. M. (2021). Analysis and simulation of plant disease progress curves in R: introducing the epifitter package. *Phytopathology Research*, 3(1). <https://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., & Del Ponte, E. M. (2021). RGB-based phenotyping of foliar disease severity under controlled conditions. *Tropical Plant Pathology*, 47(1), 105–117. <https://doi.org/10.1007/s40858-021-00448-y>
- Alves, K. S., Shah, D. A., Dillard, H. R., Del Ponte, E. M., & Pethybridge, S. J. (2022, August 22). *From reanalysis data to inference: a framework for linking environment to plant disease epidemics at the regional scale*. <http://dx.doi.org/10.31219/osf.io/9tgau>
- Analytis, S. (1977). Über die Relation zwischen biologischer Entwicklung und Temperatur bei phytopathogenen Pilzen. *Journal of Phytopathology*, 90(1), 64–76. <https://doi.org/10.1111/j.1439-0434.1977.tb02886.x>
- Baddeley, A., Turner, R., Møller, J., & Hazelton, M. (2005). Residual analysis for spatial point processes (with discussion). *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(5), 617–666. <https://doi.org/10.1111/j.1467-9868.2005.00519.x>
- Baddeley, A., Diggle, P. J., Hardegen, A., Lawrence, T., Milne, R. K., & Nair, G. (2014). On tests of spatial pattern based on simulation envelopes. *Ecological Monographs*, 84(3), 477–489. <https://doi.org/10.1890/13-2042.1>
- Bailey, J. E. (1999). Integrated Method of Organizing, Computing, and Deploying Weather-Based Disease Advisories for Selected Peanut Disease. *Peanut Science*, 26(2), 74–80. <https://doi.org/10.3146/i0095-3679-26-2-3>
- Barnhart, H. X., Haber, M., & Song, J. (2002). Overall Concordance Correlation Coefficient for Evaluating Agreement Among Multiple Observers. *Biometrics*, 58(4), 1020–1027. <https://doi.org/10.1111/j.0006-341x.2002.01020.x>
- Bates, D., Mächler, M., Bolker, B., & Walker, S. (2015). Fitting Linear Mixed-Effects Models Using `lme4`. *Journal of Statistical Software*, 67(1). <https://doi.org/10.18637/jss.v067.i01>

- Bernard, F., Sache, I., Suffert, F., & Chelle, M. (2013). The development of a foliar fungal pathogen does react to leaf temperature!. *New Phytologist*, 198(1), 232–240. <https://doi.org/10.1111/nph.12134>
- Best, A., & Cunliffe, N. (2026). Fitting a lattice model with local and global transmission to spread of a plant disease. *PLOS Computational Biology*, 22, e1013404.
- Bock, C. H., Chiang, K.-S., & Del Ponte, E. M. (2021b). 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(1), 25–42. <https://doi.org/10.1007/s40858-021-00439-z>
- Bock, C. H., Pethybridge, S. J., Barbedo, J. G. A., Esker, P. D., Mahlein, A.-K., & Del Ponte, E. M. (2021a). A phytopathometry glossary for the twenty-first century: towards consistency and precision in intra- and inter-disciplinary dialogues. *Tropical Plant Pathology*, 47(1), 14–24. <https://doi.org/10.1007/s40858-021-00454-0>
- Bourke, P. M. A. (1970). Use of Weather Information in the Prediction of Plant Disease Epiphytotics. *Annual Review of Phytopathology*, 8(1), 345–370. <https://doi.org/10.1146/annurev.py.08.090170.002021>
- Brasset, P., & Gilligan, C. (1988). A model for primary and secondary infection in botanical epidemics. *Journal of Plant Diseases and Protection*, 95, 352–360.
- Brooks, M. E., Kristensen, K., van Benthem, K. J., Magnusson, A., Berg, C. W., Nielsen, A., Skaug, H. J., Maechler, M., & Bolker, B. M. (2017). glmmTMB Balances Speed and Flexibility Among Packages for Zero-inflated Generalized Linear Mixed Modeling. *The R Journal*, 9(2), 378–400. <https://doi.org/10.32614/RJ-2017-066>
- Brown, V. A. (2021). An Introduction to Linear Mixed-Effects Modeling in R. *Advances in Methods and Practices in Psychological Science*, 4(1), 251524592096035. <https://doi.org/10.1177/2515245920960351>
- Bulger, M. A. (1987). Influence of Temperature and Wetness Duration on Infection of Strawberry Flowers by *Botrytis cinerea* and Disease Incidence of Fruit Originating from Infected Flowers. *Phytopathology*, 77(8), 1225. <https://doi.org/10.1094/phyto-77-1225>
- Bussell, E., & Cunliffe, N. (2022). Optimal strategies to protect a sub-population at risk due to an established epidemic. *Journal of the Royal Society: Interface*, 19, 20210718.
- Caffi, T., Rossi, V., Cossu, A., & Fronteddu, F. (2007). Empirical vs. mechanistic models for primary infections of *Plasmopara viticola**. *EPPO Bulletin*, 37(2), 261–271. <https://doi.org/10.1111/j.1365-2338.2007.01120.x>
- Caffi, T., Rossi, V., Legler, S. E., & Bugiani, R. (2010). A mechanistic model simulating ascospore infections by *Erysiphe necator*, the powdery mildew fungus of grapevine. *Plant Pathology*, 60(3), 522–531. <https://doi.org/10.1111/j.1365-3059.2010.02395.x>
- Café-Filho, A. C., Santos, G. R., & Laranjeira, F. F. (2010). Temporal and spatial dynamics of watermelon gummy stem blight epidemics. *European Journal of Plant Pathology*, 128(4), 473–482. <https://doi.org/10.1007/s10658-010-9674-1>
- Calvero Jr, S. B., Coakley, S. M., & Teng, P. S. (1996). Development of empirical forecasting models for rice blast based on weather factors. *Plant Pathology*, 45(4), 667–678. <https://doi.org/10.1046/j.1365-3059.1996.d01-168.x>

- Campbell, C. L., & Madden, L. V. (1990). *Introduction to plant disease epidemiology* (p. 532). Wiley.
- Cao, X., Yao, D., Xu, X., Zhou, Y., Ding, K., Duan, X., Fan, J., & Luo, Y. (2015). Development of Weather- and Airborne Inoculum-Based Models to Describe Disease Severity of Wheat Powdery Mildew. *Plant Disease*, 99(3), 395–400. <https://doi.org/10.1094/pdis-02-14-0201-re>
- Chester, K. S., & . (1950). *Plant disease losses : their appraisal and interpretation* /. <https://doi.org/10.5962/bhl.title.86198>
- Chiang, K.-S., & 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(1), 58–73. <https://doi.org/10.1007/s40858-021-00446-0>
- Chiang, K.-S., Chang, Y. M., Liu, H. I., Lee, J. Y., El Jarroudi, M., & Bock, C. (2023). Survival Analysis as a Basis to Test Hypotheses When Using Quantitative Ordinal Scale Disease Severity Data. *Phytopathology*®. <https://doi.org/10.1094/phyto-02-23-0055-r>
- Chiang, K.-S., Liu, S.-C., Bock, C. H., & Gottwald, T. R. (2014). What Interval Characteristics Make a Good Categorical Disease Assessment Scale?. *Phytopathology*®, 104(6), 575–585. <https://doi.org/10.1094/phyto-10-13-0279-r>
- Coakley, S. M. (1985). Model for Predicting Severity of Septoria tritici Blotch on Winter Wheat. *Phytopathology*, 75(11), 1245. <https://doi.org/10.1094/phyto-75-1245>
- Coakley, S. M. (1988). Predicting Stripe Rust Severity on Winter Wheat Using an Improved Method for Analyzing Meteorological and Rust Data. *Phytopathology*, 78(5), 543. <https://doi.org/10.1094/phyto-78-543>
- Coakley, S. M., McDaniel, L. R., & Line, R. F. (1988). Quantifying how climatic factors affect variation in plant disease severity: A general method using a new way to analyze meteorological data. *Climatic Change*, 12(1), 57–75. <https://doi.org/10.1007/bf00140264>
- Cruz, C. D., & Valent, B. (2017). Wheat blast disease: danger on the move. *Tropical Plant Pathology*, 42(3), 210–222. <https://doi.org/10.1007/s40858-017-0159-z>
- Cunniffe, N. J., & Gilligan, C. A. (2020). Use of Mathematical Models to Predict Epidemics and to Optimize Disease Detection and Management. In J. B. Ristaino & A. Records (Eds.), *Emerging Plant Diseases and Global Food Security: Emerging Plant Diseases and Global Food Security* (pp. 239–266). American Phytopathological Society Press.
- Cunniffe, N. J., Koskella, B., Metcalf, C. J. E., Parnell, S., Gottwald, T. R., & Gilligan, C. A. (2015). Thirteen challenges in modelling plant disease. *Epidemics*, 10, 6–10.
- Cunniffe, N. J., Stutt, R. J. O. H., Bosch, F. van den, & Gilligan, C. A. (2012). Time-dependent infectivity and flexible latent and infectious periods in compartmental models of plant disease. *Phytopathology*, 102, 365–380.
- Cunniffe, N., Cobb, R., Meentemeyer, R., Rizzo, D., & Gilligan, C. (2016). Modeling when, where and how to manage a forest epidemic, motivated by sudden oak death in California. *Proceedings of the National Academy of Sciences*, 113, 5640–5645.
- Cunniffe, N., Hamelin, F., Iggidr, A., Rapaport, A., & Sallet, G. (2024). *Identifiability and Observability in Epidemiological Models. A primer*. Springer.

- Cunniffe, N., Stutt, R., DeSimone, R., Gottwald, T., & Gilligan, C. (2015). Optimising and communicating options for the control of invasive plant disease when there is epidemiological uncertainty. *PLOS Computational Biology*, 11, e1004211.
- Cunniffe, N., Taylor, N., Hamelin, F., & Jeger, M. (2021). Epidemiological and ecological consequences of virus manipulation of host and vector in plant virus transmission. *PLOS Computational Biology*, 17, e1009759.
- Dalla Lana, F., Madden, L. V., & Paul, P. A. (2021a). Natural Occurrence of Maize Gibberella Ear Rot and Contamination of Grain with Mycotoxins in Association with Weather Variables. *Plant Disease*, 105(1), 114–126. <https://doi.org/10.1094/pdis-05-20-0952-re>
- Dalla Lana, F., Madden, L. V., & Paul, P. A. (2021b). Logistic Models Derived via LASSO Methods for Quantifying the Risk of Natural Contamination of Maize Grain with Deoxynivalenol. *Phytopathology*®, 111(12), 2250–2267. <https://doi.org/10.1094/phyto-03-21-0104-r>
- Dalla Lana, F., Ziegelmann, P. K., Maia, A. d. H. N., Godoy, C. V., & Del Ponte, E. M. (2015). Meta-Analysis of the Relationship Between Crop Yield and Soybean Rust Severity. *Phytopathology*®, 105(3), 307–315. <https://doi.org/10.1094/phyto-06-14-0157-r>
- Dalla Pria, M., Christiano, R. C. S., Furtado, E. L., Amorim, L., & Bergamin Filho, A. (2006). Effect of temperature and leaf wetness duration on infection of sweet oranges by Asiatic citrus canker. *Plant Pathology*, 55(5), 657–663. <https://doi.org/10.1111/j.1365-3059.2006.01393.x>
- De Cól, M., Coelho, M., & Del Ponte, E. M. (2024). Weather-Based Logistic Regression Models for Predicting Wheat Head Blast Epidemics. *Plant Disease*, 108(7), 2206–2213. <https://doi.org/10.1094/pdis-11-23-2513-re>
- De Rossi, R., Guerra, F., Plazas, M., Vuletic, E., Brücher, E., Guerra, G., & Reis, E. (2022). Crop damage, economic losses, and the economic damage threshold for northern corn leaf blight. *Crop Protection*, 154, 105901. <https://doi.org/10.1016/j.cropro.2021.105901>
- De Wolf, E. D., Madden, L. V., & Lipps, P. E. (2003). Risk Assessment Models for Wheat Fusarium Head Blight Epidemics Based on Within-Season Weather Data. *Phytopathology*®, 93(4), 428–435. <https://doi.org/10.1094/phyto.2003.93.4.428>
- De Wolf, E. D., & Isard, S. A. (2007). Disease Cycle Approach to Plant Disease Prediction. *Annual Review of Phytopathology*, 45(1), 203–220. <https://doi.org/10.1146/annurev.phyto.44.070505.143329>
- De Wolf, E. D., Andersen Onofre, K. F., & Lollato, R. P. (2023). Early Season Environmental Indicators of Wheat Stripe Rust Epidemics in Kansas and the Central Great Plains Region of the United States. *Plant Disease*, 107(7), 2119–2125. <https://doi.org/10.1094/pdis-08-22-1873-re>
- Del Ponte, E. M., Godoy, C. V., Li, X., & Yang, X. B. (2006). Predicting Severity of Asian Soybean Rust Epidemics with Empirical Rainfall Models. *Phytopathology*®, 96(7), 797–803. <https://doi.org/10.1094/phyto-96-0797>
- Del Ponte, E., Mikaberidze, A., Bebb, D., Halliday, F., & Cunniffe, N. (2026). The epidemiology of wild-crop interfaces: integrating ecology, evolution and management through modelling. *Philosophical Transactions B*, 381, 20250110.
- Del Ponte, E. M., Cazón, L. I., Alves, K. S., Pethybridge, S. J., & 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(1), 43–57. <https://doi.org/10.1007/s40858-021-00479-5>
- Del Ponte, E. M., Fernandes, J. M. C., & Pavan, W. (2005). A risk infection simulation model for fusarium head blight of wheat. *Fitopatologia Brasileira*, 30(6), 634–642. <https://doi.org/10.1590/s0100-41582005000600011>
- Del Ponte, E. M., Mahlein, A.-K., & Bock, C. H. (2024). *Plant disease quantification* (pp. 211–225). Elsevier. <https://doi.org/10.1016/b978-0-12-822429-8.00006-6>
- Del Ponte, E. M., Nelson, S. C., & Pethybridge, S. J. (2019). Evaluation of App-Embedded Disease Scales for Aiding Visual Severity Estimation of *Cercospora* Leaf Spot of Table Beet. *Plant Disease*, 103(6), 1347–1356. <https://doi.org/10.1094/pdis-10-18-1718-re>
- Del Ponte, E. M., Pethybridge, S. J., Bock, C. H., Michereff, S. J., Machado, F. J., & Spolti, P. (2017). Standard Area Diagrams for Aiding Severity Estimation: Scientometrics, Pathosystems, and Methodological Trends in the Last 25 Years. *Phytopathology*®, 107(10), 1161–1174. <https://doi.org/10.1094/phyto-02-17-0069-fi>
- Donnelly, R., Cuniffe, N., Carr, J., & Gilligan, C. (2019). Pathogenic modification of plants enhances long-distance dispersal of non-persistently transmitted viruses to new hosts. *Ecology*, 100, e2725.
- Duffeck, M. R., Santos Alves, K. dos, Machado, F. J., Esker, P. D., & Del Ponte, E. M. (2020). Modeling Yield Losses and Fungicide Profitability for Managing Fusarium Head Blight in Brazilian Spring Wheat. *Phytopathology*®, 110(2), 370–378. <https://doi.org/10.1094/phyto-04-19-0122-r>
- Duthie, J. A. (1997). Models of the Response of Foliar Parasites to the Combined Effects of Temperature and Duration of Wetness. *Phytopathology*®, 87(11), 1088–1095. <https://doi.org/10.1094/phyto.1997.87.11.1088>
- El Jarroudi, M., Kouadio, L., Bock, C. H., El Jarroudi, M., Junk, J., Pasquali, M., Maraite, H., & Delfosse, P. (2017). A Threshold-Based Weather Model for Predicting Stripe Rust Infection in Winter Wheat. *Plant Disease*, 101(5), 693–703. <https://doi.org/10.1094/pdis-12-16-1766-re>
- Elderfield, J., Lopez-Ruiz, F., Bosch, F. van den, & Cuniffe, N. (2018). Using epidemiological principles to explain fungicide resistance management tactics: why do mixtures outperform alternations?. *Phytopathology*, 108, 803–817.
- Ellis, J., Lazaro, E., Duarte, B., Magalhaes, T., Duarte, A., Benhadi-Marin, J., Pereira, J., Vicent, A., Parnell, S., & Cuniffe, N. (2025). Developing epidemiological preparedness for a plant disease invasion: modelling citrus huánglóngbīng in the European Union. *Plants, People, Planet*, 7, 1403–1423.
- Esser, D. S., Leveau, J. H., Meyer, K. M., & Wiegand, K. (2014). Spatial scales of interactions among bacteria and between bacteria and the leaf surface. *FEMS Microbiology Ecology*, 91(3). <https://doi.org/10.1093/femsec/fiu034>
- Evans, K. J. (1992). A Model Based on Temperature and Leaf Wetness Duration for Establishment of *Alternaria* Leaf Blight of Muskmelon. *Phytopathology*, 82(8), 890. <https://doi.org/10.1094/phyto-82-890>
- Fabre, F., Coville, J., & Cuniffe, N. (2021). Optimising reactive disease management using spatially explicit models at the landscape scale. In P. Scott, R. Strange, L. Korsten, & L. Gullino (Eds.), *Plant Diseases and Food Security in the 21st Century: Plant Diseases and Food Security in the 21st Century* (Chapter 4). Springer.

- Fabre, F., Rousseau, E., Mailleret, L., & Moury, B. (2012). Durable strategies to deploy plant resistance in agricultural landscapes. *New Phytologist*, 193, 1064–1075.
- Falla, E., & Cuniffe, N. (2026). The benefits of an attractive companion: modelling control of non-persistently transmitted plant viruses via trap or repellent companion plants. *Plant Pathology*, 75, e70171.
- Fedele, G., Brischetto, C., Rossi, V., & Gonzalez-Dominguez, E. (2022). A Systematic Map of the Research on Disease Modelling for Agricultural Crops Worldwide. *Plants*, 11(6), 724. <https://doi.org/10.3390/plants11060724>
- Franceschi, V. T., Alves, K. S., Mazaro, S. M., Godoy, C. V., Duarte, H. S. S., & 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(3), 495–505. <https://doi.org/10.1111/ppa.13148>
- Franci, L. J. (2001). The..Disease Triangle: A Plant Pathological Paradigm Revisited. *The Plant Health Instructor*. <https://doi.org/10.1094/phi-t-2001-0517-01>
- Gertheiss, J., Rügamer, D., Liew, B. X. W., & Greven, S. (2024). Functional Data Analysis: An Introduction and Recent Developments. *Biometrical Journal*, 66(7). <https://doi.org/10.1002/bimj.202300363>
- Gibson, G., & Austin, E. (1996). Fitting and testing spatio-temporal stochastic models with application in plant epidemiology. *Plant Pathology*, 45, 172–184.
- 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., & Costamilan, L. M. (2016). Asian soybean rust in Brazil: past, present, and future. *Pesquisa Agropecuária Brasileira*, 51(5), 407–421. <https://doi.org/10.1590/s0100-204x2016000500002>
- González-Domínguez, E., Caffi, T., Rossi, V., Salotti, I., & Fedele, G. (2023). Plant Disease Models and Forecasting: Changes in Principles and Applications over the Last 50 Years. *Phytopathology*®, 113(4), 678–693. <https://doi.org/10.1094/phyto-10-22-0362-kd>
- González-Domínguez, E., Martins, R. B., Del Ponte, E. M., Michereff, S. J., García-Jiménez, J., & 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*. <https://doi.org/10.1007/s10658-014-0400-2>
- Gouache, D., Léon, M. S., Duyme, F., & Braun, P. (2015). A novel solution to the variable selection problem in Window Pane approaches of plant pathogen – Climate models: Development, evaluation and application of a climatological model for brown rust of wheat. *Agricultural and Forest Meteorology*, 205, 51–59. <https://doi.org/10.1016/j.agrformet.2015.02.013>
- Gubbins, S., & Gilligan, C. (1997). Biological control in a disturbed environment. *Philosophical Transactions of the Royal Society of London, B*, 352, 1935–1949.
- Hamelin, F., Allen, L., Bokil, V., Gross, L., Hilker, F., Jeger, M., Manore, C., Power, A., Rúa, M., & Cuniffe, N. (2019). Coinfections by noninteracting pathogens are not independent and require new tests of interaction. *PLOS Biology*, 17, e300551.
- Hamelin, F., Castel, M., Poggi, S., Andrivon, D., & Mailleret, L. (2011). Seasonality and the evolutionary divergence of plant parasites. *Ecology*, 92, 2159–2166.

- Hank, T. B., Berger, K., Bach, H., Clevers, J. G. P. W., Gitelson, A., Zarco-Tejada, P., & Mauser, W. (2018). Spaceborne Imaging Spectroscopy for Sustainable Agriculture: Contributions and Challenges. *Surveys in Geophysics*, 40(3), 515–551. <https://doi.org/10.1007/s10712-018-9492-0>
- Harikrishnan, R., & Río, L. E. d. (2008). A Logistic Regression Model for Predicting Risk of White Mold Incidence on Dry Bean in North Dakota. *Plant Disease*, 92(1), 42–46. <https://doi.org/10.1094/pdis-92-1-0042>
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer New York. <https://doi.org/10.1007/978-0-387-84858-7>
- Hau, B., & Kranz, J. (1990). *Mathematics and Statistics for Analyses in Epidemiology* (pp. 12–52). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-75398-5_2
- Hebert, T. T. (1982). The Rationale for the Horsfall-Barratt Plant Disease Assessment Scale. *Phytopathology*, 72(10), 1269. <https://doi.org/10.1094/phyto-72-1269>
- Hjelkrem, A.-G. R., Aamot, H. U., Lillemo, M., Sørensen, E. S., Brodal, G., Russenes, A. L., Edwards, S. G., & Hofgaard, I. S. (2021). Weather Patterns Associated with DON Levels in Norwegian Spring Oat Grain: A Functional Data Approach. *Plants*, 11(1), 73. <https://doi.org/10.3390/plants11010073>
- Hughes, G., & Madden, L. V. (1992). Aggregation and incidence of disease. *Plant Pathology*, 41(6), 657–660. <https://doi.org/10.1111/j.1365-3059.1992.tb02549.x>
- Huichun YE, Senzheng CHEN, Anting GUO, Chaojia NIE, & Jingjing WANG. (2022, December 30). *A dataset of UAV multispectral images for banana Fusarium wilt survey*. Science Data Bank. <https://doi.org/10.57760/SCIENCEDB.07000>
- Hyatt-Twynam, S., Parnell, S., Stutt, R., Gottwald, T., Gilligan, C., & Cuniffe, N. (2017). Risk-based management of invading plant disease. *New Phytologist*, 214, 1317–1329.
- Islam, M. T., Kim, K.-H., & Choi, J. (2019). Wheat Blast in Bangladesh: The Current Situation and Future Impacts. *The Plant Pathology Journal*, 35(1), 1–10. <https://doi.org/10.5423/ppj.rw.08.2018.0168>
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning with Applications in R* (2nd ed.). Springer.
- Jeger, M. J., & 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(1), 32–40. <https://doi.org/10.1007/s001220051615>
- Jesus Junior, W. C. d., & Bassanezi, R. B. (2004). Análise da dinâmica e estrutura de focos da morte súbita dos citros. *Fitopatologia Brasileira*, 29(4), 399–405. <https://doi.org/10.1590/s0100-41582004000400007>
- Ji, T., Languasco, L., Li, M., & Rossi, V. (2021). Effects of Temperature and Wetness Duration on Infection by *Coniella diplodiella*, the Fungus Causing White Rot of Grape Berries. *Plants*, 10(8), 1696. <https://doi.org/10.3390/plants10081696>
- Ji, T., Salotti, I., Altieri, V., Li, M., & Rossi, V. (2023). Temperature-Dependent Growth and Spore Germination of Fungi Causing Grapevine Trunk Diseases: Quantitative Analysis of Literature Data. *Plant Disease*, 107(5), 1386–1398. <https://doi.org/10.1094/pdis-09-22-2249-re>

- Jones, H. G., & Vaughan, R. A. (2010). *Remote Sensing of Vegetation: Principles, Techniques, and Applications*. Oxford University Press.
- Keeling, M. J., & Rohani, P. (2008). *Modeling Infectious Diseases in Humans and Animals*. Princeton University Press.
- Kouadio, L., El Jarroudi, M., Belabess, Z., Laasli, S.-E., Roni, M. Z. K., Amine, I. D. I., Mokhtari, N., Mokrini, F., Junk, J., & Lahlali, R. (2023). A Review on UAV-Based Applications for Plant Disease Detection and Monitoring. *Remote Sensing*, 15(17), 4273. <https://doi.org/10.3390/rs15174273>
- Krause, R. A., & Massie, L. B. (1975). Predictive Systems: Modern Approaches to Disease Control. *Annual Review of Phytopathology*, 13(1), 31–47. <https://doi.org/10.1146/annurev.py.13.090175.000335>
- Krause, R., Massie, L., & Hyre, R. (1975). BLITECAST: A Computerized Forecast of Potato Late Blight. *The Plant Disease Reporter*, 59(2), 95.
- Kriss, A. B., Paul, P. A., & Madden, L. V. (2010). Relationship Between Yearly Fluctuations in Fusarium Head Blight Intensity and Environmental Variables: A Window-Pane Analysis. *Phytopathology*, 100(8), 784–797. <https://doi.org/10.1094/phyto-100-8-0784>
- Lannou, C. (2012). Variation and Selection of Quantitative Traits in Plant Pathogens. *Annual Review of Phytopathology*, 50(1), 319–338. <https://doi.org/10.1146/annurev-phyto-081211-173031>
- Laranjeira, F., Bergamin Filho, A., & Amorim, L. (1998). Dinâmica e estrutura de focos da clorose variegada dos citros (CVC). *Fitopatologia Brasileira*, 23(1), 36–41.
- Laranjeira, F. F., Bergamin Filho, A., Amorim, L., & Gottwald, T. R. (2004). Dinâmica espacial da clorose variegada dos citros em três regiões do Estado de São Paulo. *Fitopatologia Brasileira*, 29(1), 56–65. <https://doi.org/10.1590/s0100-41582004000100009>
- Lehner, M. S., Pethybridge, S. J., Meyer, M. C., & 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(3), 460–468. <https://doi.org/10.1111/ppa.12590>
- Leiminger, J. H., & Hausladen, H. (2012). Early Blight Control in Potato Using Disease-Orientated Threshold Values. *Plant Disease*, 96(1), 124–130. <https://doi.org/10.1094/pdis-05-11-0431>
- Li, B., Madden, L. V., & Xu, X. (2011). Spatial analysis by distance indices: an alternative local clustering index for studying spatial patterns. *Methods in Ecology and Evolution*, 3(2), 368–377. <https://doi.org/10.1111/j.2041-210x.2011.00165.x>
- Li, F., Upadhyaya, N. M., Sperschneider, J., Matny, O., Nguyen-Phuc, H., Mago, R., Raley, C., Miller, M. E., Silverstein, K. A. T., Henningsen, E., Hirsch, C. D., Visser, B., Pretorius, Z. A., Steffenson, B. J., Schwessinger, B., Dodds, P. N., & Figueroa, M. (2019). Emergence of the Ug99 lineage of the wheat stem rust pathogen through somatic hybridisation. *Nature Communications*, 10(1). <https://doi.org/10.1038/s41467-019-12927-7>
- Lin, L. I.-K. (1989). A Concordance Correlation Coefficient to Evaluate Reproducibility. *Biometrics*, 45(1), 255. <https://doi.org/10.2307/2532051>
- Liu, H. I., Tsai, J. R., Chung, W. H., Bock, C. H., & Chiang, K. S. (2019). Effects of Quantitative Ordinal Scale Design on the Accuracy of Estimates of Mean Disease Severity. *Agronomy*, 9(9), 565. <https://doi.org/10.3390/agronomy9090565>

- Lowe, A., Harrison, N., & French, A. P. (2017). Hyperspectral image analysis techniques for the detection and classification of the early onset of plant disease and stress. *Plant Methods*, 13(1). <https://doi.org/10.1186/s13007-017-0233-z>
- MacHardy, W. E. (1989). A Revision of Mills's Criteria for Predicting Apple Scab Infection Periods. *Phytopathology*, 79(3), 304. <https://doi.org/10.1094/phyto-79-304>
- Madden, L. (1978). FAST, a Forecast System for Alternaria solani on Tomato. *Phytopathology*, 68(9), 1354. <https://doi.org/10.1094/phyto-68-1354>
- Madden, L. V. (1982). Evaluation of Tests for Randomness of Infected Plants. *Phytopathology*, 72(2), 195. <https://doi.org/10.1094/phyto-72-195>
- Madden, L. V., & Paul, P. A. (2009). Assessing Heterogeneity in the Relationship Between Wheat Yield and Fusarium Head Blight Intensity Using Random-Coefficient Mixed Models. *Phytopathology*®, 99(7), 850–860. <https://doi.org/10.1094/phyto-99-7-0850>
- Madden, L. V., & Paul, P. A. (2011). Meta-Analysis for Evidence Synthesis in Plant Pathology: An Overview. *Phytopathology*®, 101(1), 16–30. <https://doi.org/10.1094/phyto-03-10-0069>
- Madden, L. V., Hughes, G., Moraes, W. B., Xu, X.-M., & Turechek, W. W. (2018). Twenty-Five Years of the Binary Power Law for Characterizing Heterogeneity of Disease Incidence. *Phytopathology*®, 108(6), 656–680. <https://doi.org/10.1094/phyto-07-17-0234-rvw>
- Madden, L. (2025). Reflections on the past, present, and future of quantitative plant disease epidemiology. *Annual Review of Phytopathology*, 63, 1–22.
- Madden, L., & van den Bosch, F. (2002). A population-dynamics approach to assess the threat of plant pathogens as biological weapons against annual crops. *Bioscience*, 52, 65–74.
- Madden, L. V., Esker, P. D., & Pethybridge, S. J. (2021). Forrest W. Nutter, Jr.: a career in phytopathometry. *Tropical Plant Pathology*, 47(1), 5–13. <https://doi.org/10.1007/s40858-021-00469-7>
- Madden, L. V., Hughes, G., & Bosch, F. van den (Eds.). (2007). *CHAPTER 12: Epidemics and Crop Yield* (pp. 353–388). The American Phytopathological Society. <https://doi.org/10.1094/9780890545058.012>
- Madden, L. V., Hughes, G., & van den Bosch, F. (2007c). *Spatial Aspects of Epidemics—III: Patterns of Plant Disease* (No. 9; pp. 235–278). The American Phytopathological Society. <https://doi.org/10.1094/9780890545058.009>
- Madden, L. V., Hughes, G., & van den Bosch, F. (2007b). *Temporal Analysis I: Quantifying and Comparing Epidemics* (No. 4; pp. 63–116). The American Phytopathological Society. <https://doi.org/10.1094/9780890545058.004>
- Madden, L. V., Hughes, G., & van den Bosch, F. (2007a). *The Study of Plant Disease Epidemics*. The American Phytopathological Society. <https://doi.org/10.1094/9780890545058>
- Magarey, R. D., & Sutton, T. B. (2007). *How to Create and Deploy Infection Models for Plant Pathogens* (pp. 3–25). Springer Netherlands. https://doi.org/10.1007/978-1-4020-6061-8_1
- Magarey, R. D., Sutton, T. B., & Thayer, C. L. (2005). A Simple Generic Infection Model for Foliar Fungal Plant Pathogens. *Phytopathology*®, 95(1), 92–100. <https://doi.org/10.1094/phyto-95-0092>

- Magarey, R. D., Travis, J. W., Russo, J. M., Seem, R. C., & Magarey, P. A. (2002). Decision Support Systems: Quenching the Thirst. *Plant Disease*, 86(1), 4–14. <https://doi.org/10.1094/pdis.2002.86.1.4>
- Malaker, P. K., Barma, N. C. D., Tiwari, T. P., Collis, W. J., Duveiller, E., Singh, P. K., Joshi, A. K., Singh, R. P., Braun, H.-J., Peterson, G. L., Pedley, K. F., Farman, M. L., & Valent, B. (2016). First Report of Wheat Blast Caused by *Magnaporthe oryzae* Pathotype *tritricum* in Bangladesh. *Plant Disease*, 100(11), 2330. <https://doi.org/10.1094/pdis-05-16-0666-pdn>
- Mehra, L. K., Cowger, C., & Ojiambo, P. S. (2017). A Model for Predicting Onset of *Stagonospora nodorum* Blotch in Winter Wheat Based on Preplanting and Weather Factors. *Phytopathology*®, 107(6), 635–644. <https://doi.org/10.1094/phyto-03-16-0133-r>
- Mehra, L. K., Cowger, C., Gross, K., & Ojiambo, P. S. (2016). Predicting Pre-planting Risk of *Stagonospora nodorum* blotch in Winter Wheat Using Machine Learning Models. *Frontiers in Plant Science*, 7. <https://doi.org/10.3389/fpls.2016.00390>
- Mikaberidze, A., Mundt, C. C., & Bonhoeffer, S. (2015,). *Data from: Invasiveness of plant pathogens depends on the spatial scale of host distribution*. Dryad. <https://doi.org/10.5061/DRYAD.F2J8S>
- Mila, A. L., Carriquiry, A. L., & Yang, X. B. (2004). Logistic Regression Modeling of Prevalence of Soybean Sclerotinia Stem Rot in the North-Central Region of the United States. *Phytopathology*®, 94(1), 102–110. <https://doi.org/10.1094/phyto.2004.94.1.102>
- Mills, W. D. (1944). Efficient use of sulfur dusts and sprays during rain to control apple scab. *Cornell Extension Bulletin*, 630.
- Molnar, C. (2019). *Interpretable Machine Learning*. Leanpub.
- Moran, P. A. P. (1950). Notes on Continuous Stochastic Phenomena. *Biometrika*, 37(1/2), 17. <https://doi.org/10.2307/2332142>
- Moreira, R. R., Silva Silveira Duarte, H. da, & 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(3), 975–982. <https://doi.org/10.1007/s10658-018-01610-0>
- Mumford, J. D., & Norton, G. A. (1984). Economics of Decision Making in Pest Management. *Annual Review of Entomology*, 29(1), 157–174. <https://doi.org/10.1146/annurev.en.29.010184.001105>
- Mundt, C. C., Ahmed, H. U., Finckh, M. R., Nieva, L. P., & Alfonso, R. F. (1999). Primary Disease Gradients of Bacterial Blight of Rice. *Phytopathology*®, 89(1), 64–67. <https://doi.org/10.1094/phyto.1999.89.1.64>
- Murray-Watson, R., & Cunliffe, N. (2023). Expanding growers' choice of disease management options can promote suboptimal social outcomes. *Plant Pathology*, 72, 933–950.
- Nelson, S. (1996). A simple analysis of disease foci. *Phytopathology*, 86(4), 432–439.
- Nutter, F. W., & Esker, P. D. (2006). The Role of Psychophysics in Phytopathology: The Weber–Fechner Law Revisited. *European Journal of Plant Pathology*, 114(2), 199–213. <https://doi.org/10.1007/s10658-005-4732-9>
- Nutter, F. W., Esker, P. D., & 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(1), 95–103. <https://doi.org/10.1007/s10658-005-1230-z>
- Nutter, F., Teng, P., & Royer, M. (1993). Terms and Concepts for Yield, Crop Loss, and Disease Thresholds. *Plant Disease*, 77, 193–211. <http://doi.org/10.1094/PD-77-211>
- Oerke, E.-C. (2020). Remote Sensing of Diseases. *Annual Review of Phytopathology*, 58(1), 225–252. <https://doi.org/10.1146/annurev-phyto-010820-012832>
- Olivoto, T. (2022). Lights, camera, pliman! An R package for plant image analysis. *Methods in Ecology and Evolution*, 13(4), 789–798. <https://doi.org/10.1111/2041-210x.13803>
- Olivoto, T., Andrade, S. M. P., & M. Del Ponte, E. (2022). Measuring plant disease severity in R: introducing and evaluating the pliman package. *Tropical Plant Pathology*, 47(1), 95–104. <https://doi.org/10.1007/s40858-021-00487-5>
- Onofri, A., Piepho, H.-P., & Kozak, M. (2018). Analysing censored data in agricultural research: A review with examples and software tips. *Annals of Applied Biology*, 174(1), 3–13. <https://doi.org/10.1111/aab.12477>
- Parker, S. K., Nutter, F. W., & Gleason, M. L. (1997). Directional Spread of Septoria Leaf Spot in Tomato Rows. *Plant Disease*, 81(3), 272–276. <https://doi.org/10.1094/pdis.1997.81.3.272>
- Paul, P. A., & Munkvold, G. P. (2004). A Model-Based Approach to Preplanting Risk Assessment for Gray Leaf Spot of Maize. *Phytopathology*®, 94(12), 1350–1357. <https://doi.org/10.1094/phyto.2004.94.12.1350>
- Pedigo, L. P., Hutchins, S. H., & Higley, L. G. (1986). Economic Injury Levels in Theory and Practice. *Annual Review of Entomology*, 31(1), 341–368. <https://doi.org/10.1146/annurev.en.31.010186.002013>
- Pegg, K. G., Coates, L. M., O'Neill, W. T., & Turner, D. W. (2019). The Epidemiology of Fusarium Wilt of Banana. *Frontiers in Plant Science*, 10. <https://doi.org/10.3389/fpls.2019.01395>
- Pereira, W. E. L., Andrade, S. M. P. de, Del Ponte, E. M., Esteves, M. B., Canale, M. C., Takita, M. A., Coletta-Filho, H. D., & De Souza, A. A. (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(6), 710–722. <https://doi.org/10.1007/s40858-020-00401-5>
- Pfender, W. F. (2003). Prediction of Stem Rust Infection Favorability, by Means of Degree-Hour Wetness Duration, for Perennial Ryegrass Seed Crops. *Phytopathology*®, 93(4), 467–477. <https://doi.org/10.1094/phyto.2003.93.4.467>
- Pietravalle, S., Shaw, M. W., Parker, S. R., & Bosch, F. van den. (2003). Modeling of Relationships Between Weather and *Septoria tritici* Epidemics on Winter Wheat: A Critical Approach. *Phytopathology*®, 93(10), 1329–1339. <https://doi.org/10.1094/phyto.2003.93.10.1329>
- Pini, A., & Vantini, S. (2016). The Interval Testing Procedure: A General Framework for Inference in Functional Data Analysis. *Biometrics*, 72(3), 835–845. <https://doi.org/10.1111/biom.12476>
- Ramsay, J. O., & Silverman, B. W. (2005). *Functional Data Analysis* (2nd ed.). Springer.
- Reis, E. M., Hoffmann, L. L., & Blum, M. (2002). Modelo de ponto crítico para estimar os danos causados pelo oídio em cevada. *Fitopatologia Brasileira*, 27, 644–646.

- Romero, J., Moral, J., Gonzalez-Dominguez, E., Agustí-Brisach, C., Roca, L. F., Rossi, V., & Trapero, A. (2021). Logistic models to predict olive anthracnose under field conditions. *Crop Protection*, 148, 105714. <https://doi.org/10.1016/j.cropro.2021.105714>
- Rossi, V., Caffi, T., Giosuè, S., & Bugiani, R. (2008). A mechanistic model simulating primary infections of downy mildew in grapevine. *Ecological Modelling*, 212(3–4), 480–491. <https://doi.org/10.1016/j.ecolmodel.2007.10.046>
- Rossi, V., Giosuè, S., & Caffi, T. (2010). *Modelling Plant Diseases for Decision Making in Crop Protection* (pp. 241–258). Springer Netherlands. https://doi.org/10.1007/978-90-481-9277-9_15
- Rossi, V., Onesti, G., Legler, S. E., & Caffi, T. (2014). Use of systems analysis to develop plant disease models based on literature data: grape black-rot as a case-study. *European Journal of Plant Pathology*, 141(3), 427–444. <https://doi.org/10.1007/s10658-014-0553-z>
- Rotem, J. (1988). *Techniques of Controlled-Condition Experiments* (pp. 19–31). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-95534-1_3
- Russell, R., & Cuniffe, N. (2025). Optimal control prevents itself from eradicating stochastic disease epidemics. *PLOS Computational Biology*, 21, e1012781.
- Sackett, K. E., & Mundt, C. C. (2005). Primary Disease Gradients of Wheat Stripe Rust in Large Field Plots. *Phytopathology*®, 95(9), 983–991. <https://doi.org/10.1094/phyto-95-0983>
- Saif, M. S., Chancia, R., Pethybridge, S., Murphy, S. P., Hassanzadeh, A., & Aardt, J. van. (2023). Forecasting Table Beet Root Yield Using Spectral and Textural Features from Hyperspectral UAS Imagery. *Remote Sensing*, 15(3), 794. <https://doi.org/10.3390/rs15030794>
- Saif, M. S., Chancia, R., Sharma, P., Murphy, S., Raqueno, N., Bauch, T., Pethybridge, S., & Aardt, J. van. (2024, August 13). *Data for: Estimation of Cercospora Leaf Spot Disease Severity in Table Beets from UAS Multispectral Images*. Mendeley Data. <https://doi.org/10.17632/V9B7RWRWX9.1>
- Salotti, I., & Rossi, V. (2023). A Mechanistic Model Accounting for the Effect of Soil Moisture, Weather, and Host Growth Stage on the Development of *Sclerotinia sclerotiorum*. *Plant Disease*, 107(2), 514–533. <https://doi.org/10.1094/pdis-12-21-2743-re>
- Salotti, I., Bove, F., & Rossi, V. (2022). Development and Validation of a Mechanistic, Weather-Based Model for Predicting Puccinia graminis f. sp. tritici Infections and Stem Rust Progress in Wheat. *Frontiers in Plant Science*, 13. <https://doi.org/10.3389/fpls.2022.897680>
- Savary, S., Nelson, A. D., Djurle, A., Esker, P. D., Sparks, A., Amorim, L., Bergamin Filho, A., Caffi, T., Castilla, N., Garrett, K., McRoberts, N., Rossi, V., Yuen, J., & Willocquet, L. (2018). Concepts, approaches, and avenues for modelling crop health and crop losses. *European Journal of Agronomy*, 100, 4–18. <https://doi.org/10.1016/j.eja.2018.04.003>
- Savary, S., Teng, P. S., Willocquet, L., & Nutter, F. W. (2006). Quantification and Modeling of Crop Losses: A Review of Purposes. *Annual Review of Phytopathology*, 44(1), 89–112. <https://doi.org/10.1146/annurev.phyto.44.070505.143342>
- Savary, S., Willocquet, L., Pethybridge, S. J., Esker, P., McRoberts, N., & Nelson, A. (2019). The global burden of pathogens and pests on major food crops. *Nature Ecology & Evolution*, 3(3), 430–439. <https://doi.org/10.1038/s41559-018-0793-y>
- Scott, P. R., & Hollins, T. W. (1974). Effects of eyespot on the yield of winter wheat. *Annals of Applied Biology*, 78(3), 269–279. <https://doi.org/10.1111/j.1744-7348.1974.tb01506.x>

- Seem, R. C. (1984). Simple Decision Aids for Practical Control of Pests. *Plant Disease*, 68(8), 656. <https://doi.org/10.1094/pd-69-656>
- Segarra, J., Jeger, M. J., & van den Bosch, F. (2001). Epidemic dynamics and patterns of plant disease. *Phytopathology*, 91, 1001–1010.
- Shah, D. A., & Madden, L. V. (2004). Nonparametric Analysis of Ordinal Data in Designed Factorial Experiments. *Phytopathology*®, 94(1), 33–43. <https://doi.org/10.1094/phyto.2004.94.1.33>
- Shah, D. A., De Wolf, E. D., Paul, P. A., & Madden, L. V. (2019b). Functional Data Analysis of Weather Variables Linked to Fusarium Head Blight Epidemics in the United States. *Phytopathology*®, 109(1), 96–110. <https://doi.org/10.1094/phyto-11-17-0386-r>
- Shah, D. A., Molineros, J. E., Paul, P. A., Willyerd, K. T., Madden, L. V., & De Wolf, E. D. (2013). Predicting Fusarium Head Blight Epidemics With Weather-Driven Pre- and Post-Anthesis Logistic Regression Models. *Phytopathology*®, 103(9), 906–919. <https://doi.org/10.1094/phyto-11-12-0304-r>
- Shah, D. A., Paul, P. A., De Wolf, E. D., & Madden, L. V. (2019a). Predicting plant disease epidemics from functionally represented weather series. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 374(1775), 20180273. <https://doi.org/10.1098/rstb.2018.0273>
- Shrout, P. E., & Fleiss, J. L. (1979). Intraclass correlations: Uses in assessing rater reliability. *Psychological Bulletin*, 86(2), 420–428. <https://doi.org/10.1037/0033-2909.86.2.420>
- Simko, I., & Piepho, H.-P. (2012). The Area Under the Disease Progress Stairs: Calculation, Advantage, and Application. *Phytopathology*®, 102(4), 381–389. <https://doi.org/10.1094/phyto-07-11-0216>
- Skaracis, G. N., Pavli, O. I., & Biancardi, E. (2010). Cercospora Leaf Spot Disease of Sugar Beet. *Sugar Tech*, 12(3–4), 220–228. <https://doi.org/10.1007/s12355-010-0055-z>
- Tan, W., Li, K., Liu, D., & Xing, W. (2023). Cercospora leaf spot disease of sugar beet. *Plant Signaling & Behavior*, 18(1). <https://doi.org/10.1080/15592324.2023.2214765>
- Taylor, N., & Cunliffe, N. (2023). Optimal resistance management for mixtures of high-risk fungicides: robustness to the initial frequency of resistance and pathogen sexual reproduction. *Phytopathology*, 113, 55–69.
- Te Beest, D. E., Paveley, N. D., Shaw, M. W., & Bosch, F. van den. (2008). Disease–Weather Relationships for Powdery Mildew and Yellow Rust on Winter Wheat. *Phytopathology*®, 98(5), 609–617. <https://doi.org/10.1094/phyto-98-5-0609>
- Tembo, B., Mulenga, R. M., Sichilima, S., M'siska, K. K., Mwale, M., Chikoti, P. C., Singh, P. K., He, X., Pedley, K. F., Peterson, G. L., Singh, R. P., & Braun, H. J. (2020). Detection and characterization of fungus (*Magnaporthe oryzae* pathotype Triticum) causing wheat blast disease on rain-fed grown wheat (*Triticum aestivum* L.) in Zambia. *PLOS ONE*, 15(9), e238724. <https://doi.org/10.1371/journal.pone.0238724>
- Thompson, R., Gilligan, C., & Cunliffe, N. (2018). Control fast or control smart: when should invading pathogens be controlled?. *PLOS Computational Biology*, 14, e1006014.
- Thompson, R., Gilligan, C., & Cunliffe, N. (2020). Will an outbreak exceed available resources for control? Estimating the risk from invading pathogens using practical definitions of a severe epidemic. *Journal of the Royal Society: Interface*, 17, 20200690.

- Thresh, J. M. (1998). In memory of James Edward Vanderplank 1909–1997. *Plant Pathology*, 47(2), 114–115. <https://doi.org/10.1046/j.1365-3059.2998.00220.x>
- van der Plank, J. E. (1963). *Plant Diseases: Epidemics and Control*. Academic Press.
- Vanderplank, J. (1963). *Plant Disease Epidemics and Control*. Elsevier. <https://doi.org/10.1016/c2013-0-11642-x>
- Viechtbauer, W. (2010). Conducting Meta-Analyses in R with the metafor Package. *Journal of Statistical Software*, 36(3). <https://doi.org/10.18637/jss.v036.i03>
- Vincent, E., Hill, E., & Parnell, S. (2025). Modelling the effectiveness of Integrated Pest Management strategies for the control of Septoria tritici blotch. *PLOS Computational Biology*, 21, e1013352.
- Wallin, J. R. (1962). Summary of recent progress in predicting late blight epidemics in United States and Canada. *American Potato Journal*, 39(8), 306–312. <https://doi.org/10.1007/bf02862155>
- Wiegand, T., & A. Moloney, K. (2004). Rings, circles, and null-models for point pattern analysis in ecology. *Oikos*, 104(2), 209–229. <https://doi.org/10.1111/j.0030-1299.2004.12497.x>
- Willbur, J. F., Fall, M. L., Bloomingdale, C., Byrne, A. M., Chapman, S. A., Isard, S. A., Magarey, R. D., McCaghey, M. M., Mueller, B. D., Russo, J. M., Schlegel, J., Chilvers, M. I., Mueller, D. S., Kabbage, M., & Smith, D. L. (2018). Weather-Based Models for Assessing the Risk of Sclerotinia sclerotiorum Apothecial Presence in Soybean (Glycine max) Fields. *Plant Disease*, 102(1), 73–84. <https://doi.org/10.1094/pdis-04-17-0504-re>
- Wilocquet, L., Savary, S., McDonald, B., & Mikaberidze, A. (2020). A polyetic modelling framework for plant disease emergence. *Plant Pathology*, 69, 1630–1643.
- Windels, C. E., Lamey, H. A., Hilde, D., Widner, J., & Knudsen, T. (1998). A Cerospora Leaf Spot Model for Sugar Beet: In Practice by an Industry. *Plant Disease*, 82(7), 716–726. <https://doi.org/10.1094/pdis.1998.82.7.716>
- Xu, X.-M., & Robinson, J. D. (2000). Effects of temperature on the incubation and latent periods of hawthorn powdery mildew (Podosphaera clandestina). *Plant Pathology*, 49(6), 791–797. <https://doi.org/10.1046/j.1365-3059.2000.00520.x>
- Xu, X.-M., & Madden, L. V. (2004). Use of SADIE statistics to study spatial dynamics of plant disease epidemics. *Plant Pathology*, 53(1), 38–49. <https://doi.org/10.1111/j.1365-3059.2004.00949.x>
- Xu, X., Madden, L. V., Edwards, S. G., Doohan, F. M., Moretti, A., Hornok, L., Nicholson, P., & Ritieni, A. (2013). Developing logistic models to relate the accumulation of DON associated with Fusarium head blight to climatic conditions in Europe. *European Journal of Plant Pathology*, 137(4), 689–706. <https://doi.org/10.1007/s10658-013-0280-x>
- Yadav, N. V. S., Vos, S. M. de, Bock, C. H., & Wood, B. W. (2012). Development and validation of standard area diagrams to aid assessment of pecan scab symptoms on fruit. *Plant Pathology*, 62(2), 325–335. <https://doi.org/10.1111/j.1365-3059.2012.02641.x>
- Ye, H., Huang, W., Huang, S., Cui, B., Dong, Y., Guo, A., Ren, Y., & Jin, Y. (2020). Recognition of Banana Fusarium Wilt Based on UAV Remote Sensing. *Remote Sensing*, 12(6), 938. <https://doi.org/10.3390/rs12060938>
- Yin, X., Kropff, M. J., McLaren, G., & Visperas, R. M. (1995). A nonlinear model for crop development as a function of temperature. *Agricultural and Forest Meteorology*, 77, 1–16.

- Yorinori, J. T., Paiva, W. M., Frederick, R. D., Costamilan, L. M., Bertagnolli, P. F., Hartman, G. E., Godoy, C. V., & Nunes, J. (2005). Epidemics of Soybean Rust (*Phakopsora pachyrhizi*) in Brazil and Paraguay from 2001 to 2003. *Plant Disease*, 89(6), 675–677. <https://doi.org/10.1094/pd-89-0675>
- Zadoks, J. C., & Schein, R. D. (1988). James Edward Vanderplank: Maverick* and Innovator. *Annual Review of Phytopathology*, 26(1), 31–37. <https://doi.org/10.1146/annurev.py.26.090188.000335>
- Zadoks, J. (1974). The role of epidemiology in modern phytopathology. *Phytopathology*, 64, 918–923.
- Zaffaroni, M., Rimbaud, L., Mailleret, L., Cuniffe, N., & Bevacqua, D. (2021). Modelling interference between vectors of non-persistently transmitted plant viruses to identify effective control strategies. *PLOS Computational Biology*, 17, e1009727.